Nous remercions notre client principal M. Zahour, ainsi que M. Ramdamne et M. Taconet, pour l'aide précieuse et les bons conseils qu'ils nous ont apportés pour la réalisation du projet.

Nous remercions notre conseiller technique, M. Liégeard, pour avoir suivi notre projet tout au long de sa réalisation et pour nous avoir posé des questions qui nous ont permis d'éclaircir les objectifs et les limites du projet.

a. Contexte

Au cours de notre deuxième année d'I.U.T. informatique, il nous est demandé de réaliser un projet informatique en équipe. Nous avons donc eut à choisir parmi une liste de sujets, un projet satisfaisant le plus grand nombre de personnes du groupe de travail. M. Zahour a suggéré un sujet consistant à la création d'un programme de segmentation d'image. Dans un premier temps, nous avions choisi un sujet qui n'a pas pu être retenu pour diverses raisons. Alors, dans un second temps, nous avons choisi le sujet proposé par M. Zahour. Ce dernier est donc devenu notre client principal. En outre, nous avons des contacts avec deux autres clients, M. Taconet et M. Ramdamne. En effet, le programme sera utilisé comme outil pour l'élaboration de la thèse de M. Ramdamne. Cette thèse porte sur la reconnaissance automatique de formulaire grâce à l'analyse des différents composants de celui-ci.

b. Présentation du sujet

La principale partie de ce projet consiste a détecter tous les rectangles blancs maximaux d'un document préalablement numérisé. Ces rectangles blancs représentent l'espace vide entre les différents éléments qui composent ce document. Par exemple, l'espace entre deux paragraphes ou à plus petite échelle entre deux pixels appartenant à des caractères. Tous ces rectangles ainsi obtenus pourrons par la suite être visualisés dans l'unique but de contrôler le traitement qui s'est effectué.

Enfin les rectangles pourront être triés en fonction de leurs coordonnées. Ce tri sera utile pour nos clients car ces derniers reprendront la liste de rectangle afin d'effectuer une segmentation par rectangulation – procédé qui permet de détecter les diverses entités d'un document : caractère, mot, paragraphe, graphique, photo etc.

L'algorithme d'extraction existant déjà (code source en langage C) nous avons créé nos propres algorithmes en C++ qui seront plus rapides à l'exécution et plus fiables car prenant en compte plus de cas.

I. Thèse de Lahsen BOUKINED

L'objectif de la thèse de M Boukined est d'analyser des documents papier numérisés à l'aide d'un scanner classique. Cela consiste à détecter les différents composants de ce document (paragraphes, graphiques, images etc.) ainsi que leur localisation afin de connaître la nature du document.
Le document original aura préalablement été numérisé en niveaux de gris puis binarisé en noir et blanc.
Cette analyse se fait en deux étapes : l'extraction de rectangles puis la segmentation de l'image.

a. Extraction de rectangles

L'extraction se fait sur deux types de rectangles : blancs et noirs.
Le but est d'extraire tous les rectangles blancs maximaux représentants les zones vierges du document entre ses différents éléments. Puis l'extraction de rectangles noirs consiste à détecter et encadrer ces éléments.

b. Segmentation de l'image par rectangulation

Cette segmentation consiste à traiter les rectangles, de manière à obtenir des regroupements entre les différentes zones. Ainsi il est possible de détecter des caractères, puis des mots, puis des paragraphes, etc.

c. Rôle du projet dans cette thèse

L'algorithme d'extraction des rectangles blancs maximaux utilisé par M. Boukined a été écrit en C. Le problème de ce langage est l'utilisation de la mémoire paginée pour le stockage des entités en cours de traitement. Or la gestion de ce type de mémoire est très coûteuse en

temps pour la machine. A cela vient s'ajouter l'algorithme en lui-même qui comprend un nombre impressionnant de calculs. Il était donc intéressant de trouver un algorithme ayant un temps d'exécution plus rapide.

La solution trouvée a été l'utilisation du langage C++ qui lui n'utilise pas de mémoire paginée grâce aux instructions de contrôle *new* et *delete*. De plus l'algorithme original présentant quelques lacunes (absence de certains cas) il était utile de le reprendre et de le compléter afin d'obtenir un résultat parfait : une liste de rectangles complète.

II. Analyse générale & Organisation

a. Analyse préliminaire commune

L'image à partir de laquelle on doit extraire les rectangles requière que celle-ci soit stockée en mémoire, dans une structure de données adéquate (cf. Structure de Données communes).

La préparation de l'image est gérée par un logiciel de photos classique (par exemple, Image In ou Paint Shop). Ce logiciel va binariser l'image, c'est-à-dire qu'il va passer d'une image en niveau de gris à une image en noir et blanc. L'entrée correspondra alors à l'image en noir et blanc. Ensuite, on enregistre cette image dans un fichier « .RAW », ce qui implique de connaître la hauteur et la largeur de l'image car un fichier « .RAW » lit le nombre de bits de l'image, placés dans des pixels, et les affiche les uns à la suite des autres. Pour se souvenir de la hauteur et de la largeur de l'image, on nomme le fichier image par ses coordonnées. Par exemple si on a une image de largeur 100 et de hauteur 150, on nomme le fichier correspondant « 100_150.RAW ».

Les environnements tel que MS-DOS ou Windows95 émette des restrictions sur les noms de fichier. En effet, le nom des fichiers est tronqué par ~1 à partir de six caractères. Les coordonnées de l'image sont donc très imprécises. Il est nécessaire de connaître la taille exacte avant l'ouverture d'un fichier « .RAW ».

Ensuite pour l'ouverture du fichier « .RAW » , l'utilisateur devra saisir les dimension de l'image. Le traitement d'extraction sera alors mis en œuvre afin d'obtenir, d'abord la liste texte des coordonnées des rectangles maximaux, ensuite un fichier binaire stockant la liste binaire des coordonnées des rectangles maximaux, enfin une image « .RAW » avec le dessin des rectangles optimaux.

Enfin, les tests de performance sur les algorithmes d'extractions consisteraient à calculer le temps écoulé en seconde relativisé selon la taille de l'image, un critère de complexité de l'image est aussi pris en compte, ainsi que la vitesse de cadencemment du microprocesseur opérant. Le but étant, bien sûr, de mettre en œuvre cette comparaison sur chacun des algorithmes d'extraction avec une même image, afin d'en conclure les conditions idéales d'utilisation de ces algorithmes en répétant la manipulation avec d'autres images.

b. Contraintes et limites du sujet

Une des contraintes la plus importante est de gérer la mémoire sans passer par le formatage de la mémoire en page de 64 Ko (concrètement en utilisant *new*).

En outre, le programme doit présenter un temps d'exécution raisonnable sur un PC de puissance moyenne (300 MHz) dans environnement Windows 16 ou 32bits (Win 3.x à Win 98). Le logiciel de développement nous est fourni il s'agit Turbo C++ 4.02.

La documentation de la thèse nous est aussi fournie.

Les limites du projet, bien que explicites, restent variables. Toutefois, une borne a été définie : un traitement d'extraction des rectangles optimaux devra prendre en entrée une image binarisée avec un logiciel indépendant, est produire trois types de sorties, le fichier binaire, un fichier « .RAW » résultat visualisable avec ce même logiciel et un affichage texte des coordonnées des rectangles.

c. Répartition des tâches

L'objectif du projet était de fournir un algorithme d'extraction plus performant que l'existant , nous avons donc décidé de répartir le groupe en sous-groupe de travail. Valère et Damien était chargé de développer un algorithme conformément aux indications des documents de la thèse fournie afin d'avoir une base pour comparer. Mathieu était chargé de concevoir et réaliser un algorithme d'extraction avec une mise en œuvre différente basé sur les principes de rectangulation de la thèse. Yann et Xavier qui avait une conception différente du problème était chargé de développer un autre algorithme. Soizic était chargée de réaliser un programme de visualisation du résultat des algorithmes d'extraction, ainsi que les structures de données additionnelles utilisé dans le traitement de la thèse. Des tests de performance sur les algorithmes d'extractions seraient préparés afin de sélectionner l'algorithme le plus rapide.

III. Structure de données communes

Pour effectuer les différents algorithmes , nous allons utiliser des structures de données ou classes communes qui seront inclus dans le fichier header *listrect.h*

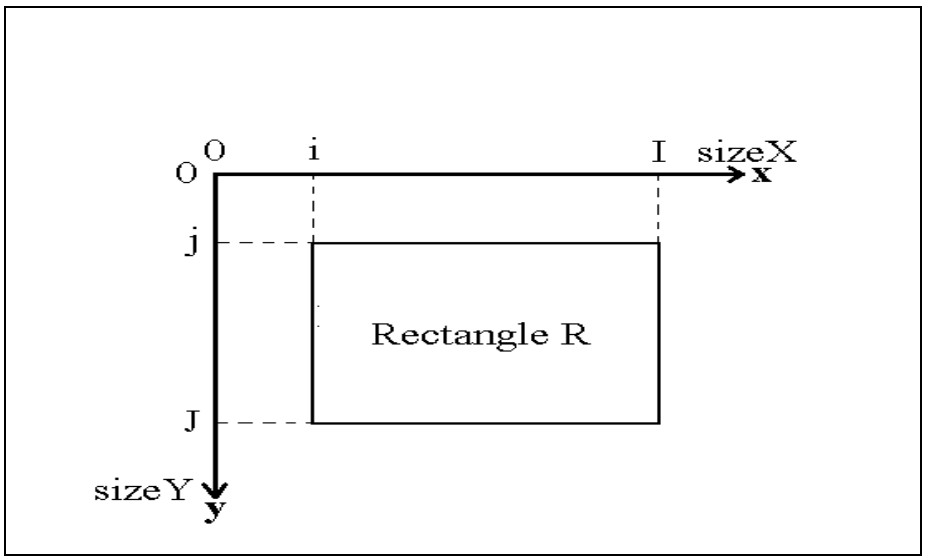
a. La classe Rectangle

- Représentation mémoire

Le rectangle sera représenté en mémoire par une suite de 4 entiers représentants ses coordonnées dans la matrice image. Cette classe ne comportera pas de méthodes complexes dans un but d'utilisation le plus simple et intuitif possible : initialisation.

- Ordre des coordonnées

Soit le rectangle R(**i,j,I,J**) :



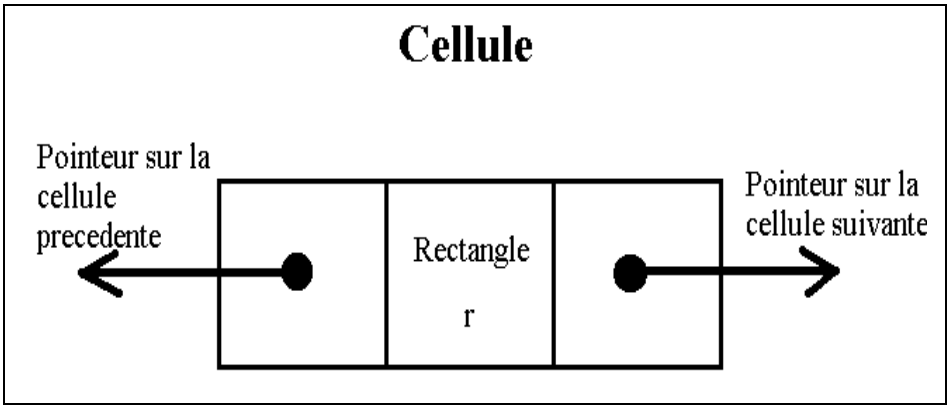
- Remarque

Un rectangle dont on ne connaît pas la fermeture voit sa coordonnée J affectée par **–1**. ainsi il aura la forme R(5,8,9,**-1**).

b. La classe Cellule (de rectangle)

- Description

La cellule est une classe annexe permettant l'introduction de rectangles dans une liste (gérée de manière dynamique). Dans ses attributs elle possède en plus d'un rectangle un pointeur sur une cellule précédente et un autre sur une cellule suivante.
Cette classe comporte des méthodes analogues à la classe rectangle permettant une transparence entre ces deux classes dans la classe liste.



- Remarque

La classe Cellule n'a pas été crée dans un but de portabilité mais uniquement dans le but de pouvoir créer une liste doublement chaînée dynamique de rectangles (ouverts ou fermés). Cette liste – décrite en 2) – sera utilisée dans l'algorithme d'extraction des rectangles blancs maximaux.

c. La classe Liste (de rectangles)

- Description

La classe ListeRectangles représente une liste de rectangles (ouverts ou fermés). Cette liste est chaînée et dynamique. Son parcours se fait grâce à un pointeur sur l'élément courant ce qui lui donne un comportement similaire à un objet Hashtable en java.

Chaque élément de la liste est une cellule (voir 2)) qui contient le rectangle a conserver et un lien vers ses cellules voisines.

- Méthodes

Cette classe contient quelques méthodes habituelles mais néanmoins nécessaires telles l'ajout, la suppression, l'affichage etc.…

Elle comporte cependant des méthodes qui nous serons utiles pour le projet telles la sauvegarde et la récupération de la liste sur fichier.

(Voir Annexe I.I Code des structures ; page)

- Remarque

Les tests sur cette liste nous ont montré qu'elle pouvait supporter plusieurs millions de rectangles. De plus, la gestion mémoire étant faite en c++ à l'aide de ''New'' et de ''delete'', on ne passe plus par la mémoire paginée, ce qui augmente considérablement la vitesse de traitement (principal objectif du projet).

IV. Les Algorithmes d'extractions

A Algorithme récursif (Yann et Xavier)

a. Introduction

Nous devons élaboré un algorithme se distinguant de ceux implémentés par les autres sous-groupes. Le traitement de ces derniers parcourt la feuille ligne par ligne afin d'extraire les rectangles blancs itérativement. Dans un objectif de rapidité, nous avons élaboré un algorithme récursif pour l'extraction des rectangles blancs consécutifs. Cet algorithme récursif est déclenché lors de la détection d'une suite de pixels blancs, conformément aux notations de Boukined cette suite de pixels s'appellera segment.

b.Analyse

Le début de notre traitement se fait itérativement afin de tester chaque bit de l'image. La recherche séquentielle permet de détecter le premier pixel blanc nécessaire au lancement de la recherche récursive, elle permet aussi de détecter tous les rectangles isolés . L'utilisation d'un algorithme récursif ne permet pas un parcours de l'image prédéfini. Selon l'enchaînement des rectangles consécutifs, l'enregistrement peut, par exemple, concerné un rectangle du bas de l'image malgré que la méthode se soit déclenchée suite à un segment détecté à la première ligne. Le traitement des rectangles blancs consécutifs se fait récursivement car les cas ne peuvent être dénombrés.

Une fois que tous les bits blancs consécutifs à un segment sont traités, nous reprenons le traitement séquentiel de manière à tester tous les bits de la feuille. Pour cela, avant le traitement récursif, nous stockons le point de reprise de la recherche séquentielle; ce point se situe à droite du premier segment déterminé. Ainsi à la prochaine détection d'un pixel blanc n'appartenant pas déjà à un rectangle, la méthode récursive sera exécuter de nouveau. Ceci implique que les seuls rectangles qui ne seront pas détectés seront entourés de noir. Ces derniers seront donc traités grâce au parcours séquentiel.

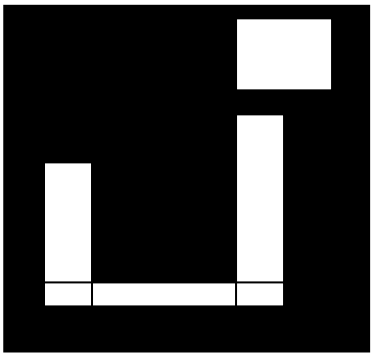
Nos différentes méthodes nous permettent d'extraire tous les rectangles. Une fois ceux-ci trouvés, il est nécessaire de les sauvegarder donc nous utilisons la liste commune composée d'une classe liste et d'une classe rectangle. L'association de ces deux classes nous permet de stocker en mémoire les rectangles pendant le traitement et, une fois ce dernier terminé, de les enregistrer dans un fichier*(enregistrement permanent sur le disque)*.

c.Conception

Afin d'extraire tous les rectangles blancs, il faut tester au moins une fois chaque pixel de l'image. Nous avons donc opter pour un parcours de haut en bas et de gauche à droite de la feuille(méthode itérative). Sur chaque bit blanc un test d'appartenance est effectué, ce test est indispensable afin d'éviter d'enregistrer plusieurs fois le même bit. Si celui-ci n'appartient à aucun rectangle, l'appel de la méthode 'detecteSegment' permet de déterminer les coordonnées du segment de pixels blancs. Il est nécessaire de mémoriser le point 'reprise du traitement itératif une fois l'exécution terminé de la méthode récursive 'traceRect'. Ce point se situe à droite du pixel noir délimitant l'extrémité du segment. La méthode récursive 'rechercheRect' va tester le bit en dessous du premier bit du segment. Si celui ci est un bit noir alors la méthode

‘fermeRect’ ajoute le segment dans la liste de rectangles fermés. Sinon la méthode ‘detecteSegment’ est de nouveau appelée. En fonction des coordonnées de ce nouveau segment, différents traitements peuvent être effectués. Si ses coordonnées sont égales à celles du segment précédent, il y a ajout d’un nouveau rectangle dans la liste de rectangles ouverts et la méthode ‘traceRect’ est appelée. Si les coordonnées sont inférieures aux coordonnées du premier rectangle, ces dernières sont complétées et ce rectangle est inséré dans la liste de rectangles fermés. Mais il faut tester s’il n’y a pas d’autres rectangles restant sous le segment (cf Exemple 2). Si une des coordonnées est supérieure, il y a prolongation du rectangle et l’ajout d’un nouveau rectangle dans la liste de rectangles ouverts. Par la suite, la méthode ‘traceRect’ test si il y a des bits au dessus du prolongement. Dans ce cas on doit détecter les coordonnées de ce segment. De toute manière les coordonnées verticaux ne dépasseront pas les coordonnées du segment initiale.

Exemple 1 :

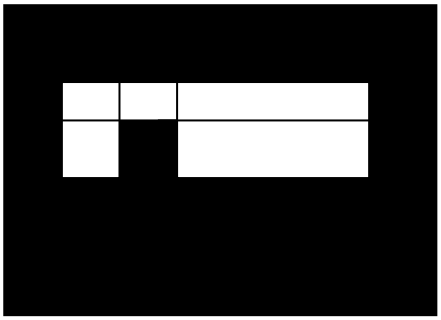


Explications de l’enchaînement des différents étapes de l’extraction.

- 1er étape : Détection du premier segment du carré situé en haut à gauche de l’image puis lancement de notre méthode récursive recherchant les segments vers le bas.
- 2ème étape : Extension du premier segment détecter jusqu’à la dernière ligne du carrée
- 3ème étape : Détection du premier segment de la colonne située à droite.
- 4ème étape : Extension de ce segment à toute la colonne.

- 5ème étape : Détection d'un segment adjacent plus grand que celui de départ => ouverture d'un nouveau rectangle. Afin de connaître i et/ou I du rectangle à ouvrir, on va rechercher la présence éventuel de zones vierges au-dessus et en-dessous
- 6ème étape : Détection d'une zone noire => fermeture du rectangle allongé horizontalement ainsi que la colonne de droite. La colonne de gauche est ouverte ; i et I ne sont pas connues donc ils sont initialisés par défaut à –1. Modifications de la valeur I de la colonne de gauche.
- 7ème étape : (méthode non implémentée) on recherche une zone vierge au dessus de celui horizontal. On exécute 'traceRectInverse' (même principe que 'traceRect' mais avec une recherche ascendante).
- 8ème étape : On détecte les pixels situé au-dessus de la colonne jusqu'à la limite de la zone vierge. Les coordonnées sont complétées et le rectangle correspondant à la colonne de droite est insérer à la liste de rectangles fermés.

Exemple 2 :



problèmes rencontres et solutions trouvées

-A la suite des premières rencontres avec le client, nous avions convenu que les différents algorithmes seraient implémenter à l'aide du langage C. L'utilisation de ce langage nous aurait obligé à concevoir une gestion de la mémoire très complexe(mémoire paginée). C'est pourquoi l'implémentation a migré du langage C au langage C++.

-Le but du projet étant de limiter le temps d'exécution, nous devons limiter le nombre d'accès disque. Donc nous pensions que nous étions dans l'impossibilité de charger la totalité du fichier image en mémoire; plusieurs solutions ont été imaginées. Après consultations du client, nous nous sommes mis d'accord sur le fait que la taille du fichier ne posait aucun problème pour la charger en mémoire. En effet, la feuille scannée ne dépasse pas le format A4 soit environ 5Mo.

-Suite à une mise en commun du travail effectué, nous nous sommes rendu compte que nous avions implémenté des sous programmes ayants le même but. Le problème était que leur conception n'était pas la même que la nôtre. Nous nous sommes donc servis de la structure de base de la liste chaînée de Valère de manière à ce que la mise en commun future soit plus facile. Nous avons complété cette classe avec de nouvelles méthodes qui sont spécifiques à notre traitement.

-Explication des raisons de l'abandon de notre algorithme

Après la répartition des tâches, nous avons approfondi la conception de notre algorithme. Afin d'obtenir une opinion objective sur sa faisabilité, nous avons contacté M. Zahour. Au cours de notre entretien, il nous a fait part de son intérêt pour l'originalité de notre conception. Cependant, il a émis des doutes sur la possibilité de dénombrer tous les cas envisageables. Nous nous sommes donc intéressés à ce problème de dénombrement. Nous pensions avoir conçu une méthode récursive assez puissante et efficace pour pouvoir déterminer tous les zones vierges consécutives à partir d'un segment donné. Afin d'alléger le corps de cette méthode, nous l'avons séparé en deux autres qui diffèrent par leur sens de recherche. Nous avons donc décidé de commencer par coder la première méthode recherchant les zones blanches situées en dessous du segment. La concrétisation de cette méthode nous a semblé interminable car plus nous progressions plus nous découvrions de nouveaux cas que nous n'avions pas prévu. Cela nous a retardé par rapport à notre planning et voyant que nous aurions du mal à finir à temps, nous avons commencé à remettre en cause l'efficacité de notre méthode. De plus certaines configurations nous ont obligés à y insérer des tests d'appartenance. Nous avons dû étendre ces tests à chaque nouveau pixel traité. Par la suite nous avons tenté d'évaluer le nombre de tests effectués et le nombre d'accès à la liste. En comparaison avec les autres algorithmes ces nombres nous ont paru trop important pour réaliser un programme ayant un temps d'exécution compétitif. Nous avons donc rencontré M. Ramdane pour lui expliquer les raisons de notre abandon. Il nous a alors proposé une solution pour éviter de parcourir à chaque fois la liste de

rectangles fermés. Malheureusement, cette proposition nous obligeait à modifier notre structure de base (remplacement du char ** tab par char *** tab, représentation de l'image en mémoire) cela impliquait la nécessité de reprendre tout notre algorithme. Cette nouvelle tâche représentait beaucoup trop de travail par rapport au temps disponible.

B] Traitement par segment (Mathieu)

a. Introduction

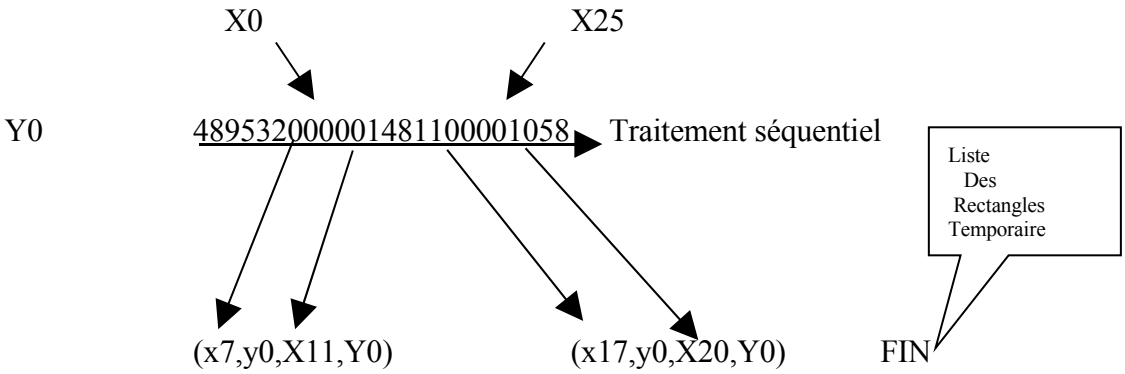
L'image à traiter est codée comme une matrice à deux dimensions afin de trouver les rectangles optimaux, il faudrait donc, dans un premier temps, détecter les segments (ou bandes) sur chaque ligne. Ainsi, dans un second temps, le traitement pourrait considérer les bandes détectées à la ligne précédente avec les bandes détectées à la ligne courante dans le but de la fusionner. Ce traitement se décompose donc en deux phases.

b. Analyse/Conception

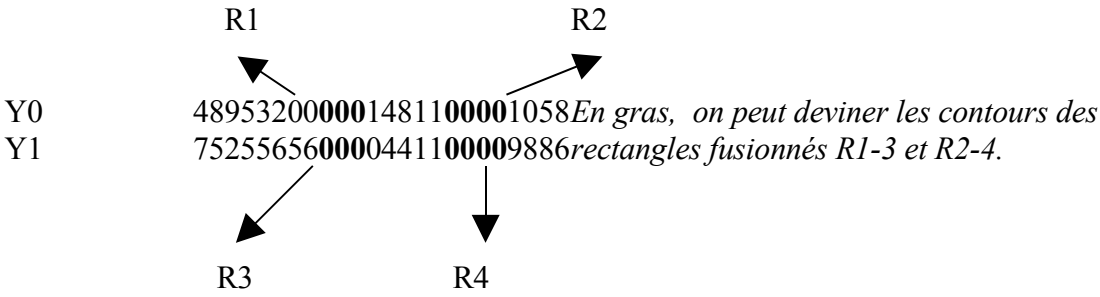
La matrice est analysée ligne par ligne : il s'agit d'un traitement séquentiel. Une image comporte Y lignes, une ligne comporte X octets. La valeur de chaque octet représente la couleur en niveau de gris (256 valeurs 0 à 255).

Le traitement comporte plusieurs étapes, d'abord il doit détecter une bande.

Lors de son parcours de la ligne, le traitement recherche la couleur à isoler (par ex : 0 couleur noire), au premier octet correspondant a cette valeur, il enregistre sa position dans la matrice (x,y) et déclare qu'une bande vient d'être ouverte après quoi il poursuit son parcours. Dés qu'il rencontre une valeur autre que la valeur à isoler, il déclare la bande comme fermée , considère sa position dans la matrice (X,Y) et enregistre la bande (x,y,X,Y) dans une liste de rectangles stockée en mémoire (Rq :une bande est un rectangles particulier).



La détection n'est qu'une étape, car il fait dégager des rectangles ayant une aire optimale sur l'image. Parallèlement, un traitement de Fusion vient analyser la possibilité de fusionner deux bandes entre elles. En fait, ce traitement intervient lorsqu'une bande est trouvée. Les bandes comparées seront assemblées si elles répondent a des critères prédéfinis de positionnement l'une par rapport à l'autre (cf. Les cas de l'Analyse - Fusion).



En gras, on peut deviner les contours des rectangles fusionnés R1-3 et R2-4.

<u>Analyse « simplifiée »pas à pas du traitement sur l'exemple :</u>

En R1 : Une bande a été détectée (R1). La liste est vide. Pas de fusion possible. Ajouter R1 dans la liste. Continuer...

En R2 : Une bande est détectée (R2). La liste contient R1. Fusion lancée. R1 et R2 incompatibles selon les cas définis. Ajouter R2 dans la liste. Continuer...

En R3 : Une bande est détectée (R3). La liste contient R1 et R2. Fusion lancée. Parcours séquentiel de la liste. R1 et R3 compatible, création du rectangle fusionné R1-3, destruction de R1 et R3, ajout dans la liste de R1-3 et des « rectangles miettes » résultant de la fusion (ici R1,R3 sont « cassés » en deux) on rajoute donc R1 et R3. R2 et R1-3 incompatibles. Continuer...

En R4 : Une bandes est détectée (R4). La liste contient R2 et R1-3. Parcours séquentiel de la liste. R2 et R4 compatibles, création du rectangle fusionné R2-4, destructions de R2 et R4, ajout dans la liste de R2-4, pas de « rectangles miettes ». Fin.

c. Cas de fusion

Il existe 8 cas lors d'une analyse fusion entre 2 rectangles ou bandes (R1 et R2). Chacun de ces cas nécessite que R1 s'ouvre avant R2 et que R1 se ferme après R2 soit (r1.Y>=r2.Y-1) et (r1.y<=r2.y-1) en C/C++.

Dans le schéma ci-dessous, 0 représente la couleur à isoler et 1 tout autre couleur, en gras le rectangle fusionné, les bandes soulignées sont les rectangles miettes:

Cas de base :

		x X	
R1	:	1**00000000**1	(r1.x==r2.x et r1.X==r2.X)
R2	:	1**00000000**1	

Cas a:

		x	X	
R1	:	1**000000**001		(r1.x==r2.x et r1.X>r2.X)
R2	:	1**000000**111		

Cas b:

		x	X	
R1	:	100**000000**1		(r1.x<r2.x et r1.X==r2.X)
R2	:	111**000000**1		

Cas c:

		x	X	
R1	:	1**0000**11111		(r1.x<r2.x et r1.X==r2.X)
R2	:	1**0000**0001		

Cas d:

		x	X	
R1	:	1111**00000**1		(r1.x>r2.x et r1.X==r2.X)
R2	:	1000**00000**1		

Cas e:

		x	X	
R1	:	1111**00**1111		(r1.x>r2.x et r1.X<r2.X)
R2	:	1000**00**001		

Cas f:

		x	X	
R1	:	0**000000000**1		(r1.x<r2.x et r1.X>r2.X)
R2	:	1**000000**11		

Cas g:

		x	X	
R1	:	11**000**0000**1**		(r1.x>r2.x et r1.X>r2.X et r2.X>=r1.x)
R2	:	00**000**11111		

Cas h:

		x	X	
R1	:	0000**00**1111		(r1.x<r2.x et r1.X<r2.X et r2.x<=r1.X)
R2	:	1111**00**000**1**		

d. Optimiser la vitesse d'exécution

L'intérêt de ce programme est, bien sûr, de fusionner un maximum de rectangles et de segments (bandes), néanmoins ces objets « empilés » dans la liste finissent par ralentir le temps d'exécution de façon alarmante, en effet pour chaque tentative de fusion, un parcours séquentiel de la liste est enclenché, or nous avons vu que deux segments ou rectangles qui fusionnent génèrent dans le pire des cas trois rectangles.

Il est donc nécessaire d'opérer un transfert des rectangles définitifs, les rectangles optimaux. Dans un premier temps, les rectangles susceptibles d'être encore fusionnables restent dans la liste temporaire (ou tampon) . Dans un second temps, lors d'un parcours de cette liste pour une fusion, un test vérifie que le rectangle est optimal et si tel est le cas, il le déplace vers la liste finale.

Schéma du mécanisme avec les même hypothèses de traitement que précédemment :

Y0	4895320**000**1481**0000**1058	*En gras, on peut deviner les contours des*
Y1	7525565**000**0441**0000**9886	*rectangles fusionnés*
Y2	969558099999900984135700	
Y3	814612179800238951111245	

Lors du parcours de Y3, on sait que les rectangles fusionnés sont définitifs. Concrétement, leur coordonnée de ligne Y (coin droit en bas) est inféreur strictement à la ligne parcourue couramment par le balayage séquentiel prinicpal.

e. Signalétique de déboggage

Cette partie explique succinctement les signalétique que j'ai adopté lors des affichages de déboggage :

\$2,3,4,5\$	*coordonnée d'une bande ou segment.*
\$2,3,4,9@	*coordonnée d'une bande forcée à la fermeture (fin de ligne).*
Cour(1,1,5,1) r(1,2,5,2) =>(1,1,5,2)	*fusion réussie avec le rectangle comparé Cour et le rectangle comparant r.*
ERR_NORM : (2,3,1,1)	*erreur coordonnée incohérente.*
[2,3,5,11] T: 18 F: 1425	*Transfert d'un rectangle optimal dans la liste finale, il y a couramment 18 bandes ou rectangles dans la liste tampon et 1425 dans la liste finale.*

Ces affichages de débogagge sont par défaut désactivé (drapeau = 0).

f. Analyse mes algorithmes principaux mis en œuvre dans le traitement

Les algorithmes seront expliqués selon le séquencement de l'exécution du programme. Ainsi, d'abord on charge en mémoire l'image du bitmap lu à partir du fichier .RAW. Ensuite, on traite l'image séquentiellement dés qu'une bande est trouvée, on lance le traite d'optimisation, qui lance le traitement de fusion. Enfin on détruit l'image mémoire.

1) Algorithme de chargement de l'image : ChargerBitmap

paramètres nécessaires : Nom du fichier , Taille image en X et en Y
Valeur de retour : un pointeur sur le tableau de pointeurs des lignes

Ouverture du fichier .RAW en mode binaire (sinon mode est ascii par défaut)
Création mémoire de l'image, il s'agit d'un tableau de pointeur qui représente les lignes pointant elles-mêmes sur un tableau de caractères (chaque caractère représente la valeur décimale de la couleur en mode 256 couleurs).On initialise au nombre de ligne Y.

Pour toute ligne Y de la matrice

Création mémoire d'un tableau de pointeur sur caractère de taille X.

Lecture d'une ligne fichier et affectation au tableau (lecture tamponnée par ligne)

Fin de pour tout

Renvoi d'un pointeur pointant sur le tableau de pointeur sur lignes

2) Algorithme de détection des bandes : DetecteurdeBande

paramètres nécessaires : Nom du fichier , Taille image en X et en Y , valeur décimale de la couleur à isoler en rectangles optimaux, la liste finale où stoker les rectangles optimaux
Valeur de retour : aucune (procédure)

On déclare une liste tampon (ou temporaire)
On déclare un fichier (un tableau de caractères, il s'agit du nom du fichier)
On déclare un booléen Bande, qui fera office de drapeau indiquant si une bande est ouverte ou non

Parcourir les lignes de la matrice

Parcourir les colonnes de la matrice

Le bit courant est de la couleur à isoler et le drapeau indique qu'aucune bande à était encore ouverte : ON A TROUVE LE DEBUT D'UNE BANDE

- Sauvegarde des coordonnées (x et y courant) indiquant le début de bande
- On indique qu'on a trouvé une bande (drapeau Bande activé)
- Ajouter au compteur de bandes détectées qu'on vient d'en trouver une nouvelle (incrémentation)

Le bit courant n'est de la couleur à isoler et le drapeau indique qu'une bande est actuellement « ouverte » : ON A TROUVE LA FIN DE LA BANDE OUVERTE

- On indique qu'on a fermé la bande, il n'y a plus de bande courante (drapeau Bande désactivé)
- On instancit un objet avec les coordonnées sauvegardées précédemment indiquant le début de la bande et les cordonnées de fin , soit les coordonnées courante
- On lance l'analyse-fusion (cf. AnalyseFusion) avec ce nouveau rectangle afin de voir si déjà on ne peux pas le fusion avec un précédent

Fin de parcours des colonnes

Si une bande est encore ouverte alors qu'on arrive en fin de ligne on force la fermeture. Et on traite ce cas de la même façon que précédemment : ON A TROUVE LA FIN D'UNE BANDE

Fin de parcours des lignes .

Affichage statistiques pour information utilisateur.

On détruit la liste des rectangles.

On lance la destruction de l'image mémoire (cf. LibéreBitmap)

3) Algorithme d'analyse et de fusion des bandes/rectangles en rectangles : AnalyseFusion

paramètres nécessaires : La liste des rectangles finale, la liste tampon, le rectangle à fusionner, la ligne courante du parcours séquentiel de DetecteurDeBande
Valeur de retour : aucune (procédure)

On replace le pointeur rectangle courant de la liste tampon (cf. Structure de données) sur le début de la liste.
On déclare un booléen indiquant la « fusionnabilité » du rectangle (il stockera une valeur de retour)
On déclare un rectangle principal qui stockera le rectangle résultant de la fusion
On déclare aussi 2 rectangles qui stockeront les morceaux (« rectangles miettes ») résultant de la fusion dans certains cas

Tant que la liste des rectangles n'est pas vide IL Y A UN COMPARANT (le rectangle pointé par le pointeur courant n'est pas égal à null)

Le rectangle courant n'est pas déjà optimal (ou « fermé ») ?

OUI

- ✓ L'ajouter dan la liste finale
- ✓ Le supprimer de la liste tampon (en fait un déplacement)
- ✓ Incrementer le compteur de fusion (servant dans ce cas principalement indicateur de visite de cette méthode)

NON

- Lancer la méthode de fusion (cf. Optimize) avec le rectangle à fusionner (le comparé) et le rectangle courant (le comparant) dans la liste tampon
- Si la valeur renvoyée par cette méthode (stockée dans le rectangle principal) indique qu'il est fusionnable :

    - On indique a un compteur qu'une fusion à lieu (incrément)

- On redéfinit le rectangle comparé avec le rectangles principal
- Si des « rectangles miettes » existent alors on les ajoutent tout de suite dans la liste tampon. (on sait qu'ils existent s'ils n'ont pas une valeur « signal » de non fusionnabilité cf. Optimize)
- On supprime enfin le rectangle comparant, il a était « absorbé » par le nouveau rectangle crée

- On indique au pointeur courant dans liste tampon de référencer « le rectangle suivant du courant »
- On ajoute le rectangle comparé qui a était redéfini ou non (si effectivement réussi à le fusionner)

Fin de parcours de la liste tampon

4) Algorithme de fusion et d'évaluation des cas : Optimize

paramètres nécessaires : le rectangle/bande à comparer, le rectangle/bande comparant
Valeur de retour : le rectangle résultant de la fusion, 2 rectangles miettes résultant de la fusion (si tel est le cas)

On déclare un rectangle signal de « non-fusionnabilité », ce rectangle qui sera renvoyé par cette fonction indiquera alors à la procédure d'analyse-fusion qu'aucune modification au rectangle d'origine
à était faite. Ce signal sera aussi utilisé pour signifier à cette procédure si les rectangles miettes ont étaient générer ou non.
On initialise le rectangle de retour comme par défaut non-fusionnable, et on fait de même pour les rectangles miettes indiquant ainsi leur inexistence par défaut.

Si les cordonnées en Y du rectangle comparant indique qu'il était « à l'intérieur » du rectangle comparé : ILS SONT PEUT-ETRE FUSIONNABLES SELON DES CAS DEFINIS

- Considérer chaque cas de fusion (cf. Cas de fusion). Si un cas est avéré alors initialiser le rectangle de retour principal comme le rectangle-fusion des rectangles d'origine . Initialiser selon le cas les rectangles miettes avec « les restes » des rectangles comparé et comparant.

- Si aucun la disposition de n'est pas répertoriée alors renvoyer le rectangle de retour inchangé depuis sa déclaration (soit le rectangle de non-fusionnabilité)

Fin de la fonction

5) Algorithme de destruction de l'image mémoire : LibéreBitmap

Pour toute les lignes Y de la matrice

Détruire le tableau de pointeur des lignes pointant sur les tableaux de caractère

Fin de parcours de lignes.

Détruire le pointeur pointant sur le tableau de pointeur des lignes désormais vide

e. Problèmes principaux et solutions trouvées

Le premier problème, auquel j'ai été confronté est les erreurs lors d'allocation de mémoire pour la matrice image. Ce problème a été résolut avec un changement de compilateur (de Borland c++ 3.0 vers Microsoft Visual c++ 5.00 Pro) et la modification des directives de compilation (mode mémoire = Huge) afin d'avoir la permission d'allouer sur toute la mémoire utilisateur.

Un oubli a également engendré un problème dans la détection des bandes, en effet les bandes encore "ouvertes" en fin de ligne ne peuvent être prolongées sur la ligne suivante, et doivent être forcée à la "fermeture".

Durant la phase de développement de l'algorithme de fusion, un rectangle qui devait être fusionné peut être disjoint, l'intersection des deux rectangles comparés (la partie fusionnée) et les "miettes" de rectangles résultantes. Je me suis rendu compte après relecture de la thèse et

entretien avec M. Ramdamne que ces "miettes" ne devaient être réduites seulement a l'aire différence entre l'aire totale des deux bandes et l'aire du rectangle fusionné; mais les miettes devaient être initialisées aux bandes à fusionner afin qu'elle puissent être utile pour une autre fusion.

Durant la phase de débugage et de tests, je me suis rendu compte que certaines bandes qui aurait du être fusionnées car elles répondaient à un cas de fusion précis, ne fusionner pas. Après maintes reprises du contenu du code des cas infructueuses, mon problème m'a rappeler un problème d'effet de bord que j'avais rencontrer en TP d'algorithmique. En effet, le problème ne concernait pas le fond mais la forme : c'est-à-dire le séquencement
des cas dans la méthode. En fait, un cas correspondait à la configuration de fusion, une solution de fusion était donc proposée, mais cette dernier correspondait désormais à un autre cas ! La solution consiste a la mise en commentaire des cas qui faisaient "effet de bord".

B] Traitement de la thèse (Valère-Damien)

Nous avons reproduit et amélioré/complété l'algorithme de la thèse. Pour faire cet algorithme, nous avons besoin de structures spécifiques: le segment, la cellule de segment et la liste de segments.

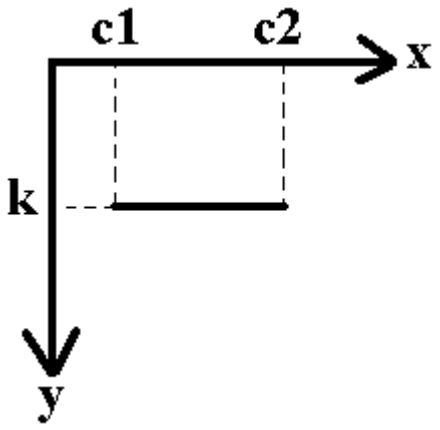
A- Structures spécifiques

1. Le Segment

- Description

Un segment est une suite en ligne d'octets consécutifs. Il est définit par ses coordonnées :

- k : la ligne ou apparaît le segment
- c1 : son abscisse d'origine
- c2 : son abscisse de fin



L'utilité d'un segment est d'ouvrir, prolonger, et dans certains cas, fermer des rectangles – ceux recherchés dans l'extraction complète des rectangles blancs maximaux.

- Remarque

Tous les segments d'une image peuvent être extraits et placés (dans l'ordre) dans une liste grâce à la fonction constuitListeSegmentd() incluse dans le fichier listsegm.h.

2. La cellule de segment : CelluleS

- Description

Tout comme la classe Cellule (cellule de rectangle), CelluleS est juste un moyen d'encapsuler un objet (un segment ici) et un pointeur pour pouvoir constituer une liste chaînée dynamique.

La différence avec ce premier type de cellules est un chaînage simple et non double.

3. La Liste de Segments

- Description

La liste de segment est utilisée pour stocker tous les segments détectés dans une image. C'est sur cette liste que nous allons effectuer l'algorithme d'extraction des rectangles blancs maximaux. En effet, c'est un segment qui ouvre, prolonge et ferme les rectangles, de plus, une fois cette liste initialisée, les manipulations mémoire nécessaires a la lecture de l'image sont terminées. Ainsi, une partie de la mémoire est économisée pour l'algorithme d'extraction seul – partie la plus coûteuse au niveau ressources.

- Remarque

De par la simplicité de la représentation d'un segment, nous aurions pu définir un segment du type struct à la place de class. Cependant, le nombre de segments dans une grande image étant impressionnant, il était logique de les stocker dans une liste chaînée, ce qui évite les copies de tableaux (très coûteux en temps d'exécution) si l'on veut garder un traitement dynamique.

B- Analyse et conception de l'algorithme

1) Introduction

Nous avons trouvé le moyen d'extraire tous les segments d'une image grâce à la fonction 'construitListeSegment'. De plus, nous savons que ce sont les segments eux-mêmes – qui combinés – vont construire et/ou prolonger et/ou fermer les rectangles blancs.

Nous appliquerons donc l'algorithme d'extraction sur la liste de segments ainsi obtenue ce qui permet de libérer des ressources mémoire et d'augmenter la lisibilité du programme.

2) Fonctionnement générale

A) Analyse

Il s'agit de comparer chaque segment à chaque rectangle ouvert précédemment. Ils se posent alors plusieurs cas de figures :

- Le segment appartient au rectangle → prolongation
- Le segment est hors limite → si le rectangle n'est pas déjà ouvert alors l'ouvrir à partir du segment
- Le segment appartient en partie au rectangle → Combinaison d'ouvertures et de fermetures de rectangles
- Dernier segments → traitement de ce segment puis fermeture de tous les rectangles ouverts.

B) Conception

DEBUT

Construire la liste de tous les segments de l'image

POUR tous les segments FAIRE

Se placer au début de la liste de rectangles ouverts

Si la liste est vide

Ajouter a la liste le rectangle correspondant au segment

FSI

Détecter si l'on est au dernier segment d'une ligne

Détecter si il y a une ligne vide entre le segment courant et le suivant

POUR tous les rectangles ouverts

Comparer les coordonnées longitudinal du segment courant avec celle du rectangle courant

Appliquer les opération d'ouverture et fermeture en fonction des 15 configurations possibles (voir Annexe).

FPT

SI on est au dernier segment de la ligne

Fermer tous les rectangles

FSI

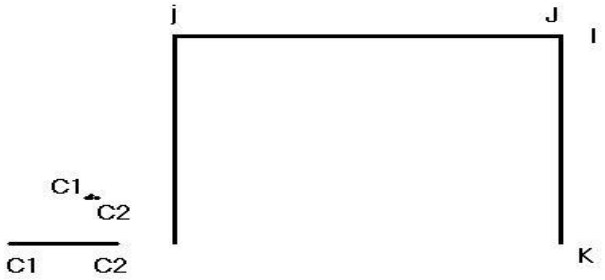
FPT

Fermer tous les rectangles

Enregistrer les rectangles fermés dans un fichier binaire

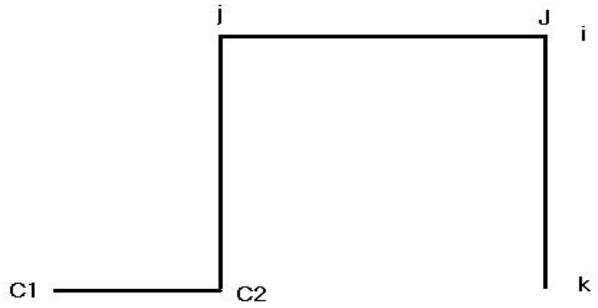
FIN

Etude et traitement des différents cas :

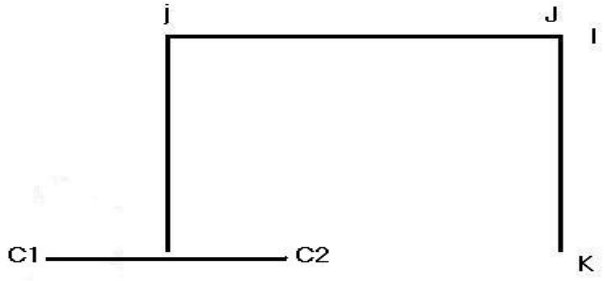


CAS A (c1<j & c2<j) : lorsque C1 et C2 du segment courant sont inférieures à j du rectangle courant de la liste des rectangles ouverts, on ajoute dans celle-ci un nouveau rectangle de coordonnées (C1, k, C2, -1). Ce cas est le même lorsque C1 et C2 sont égaux.

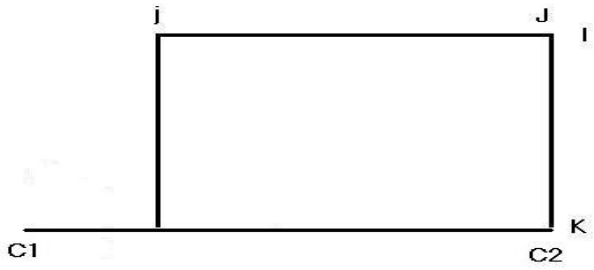
Remarque : Si le segment est le dernier de la ligne, alors il faut fermer le rectangle



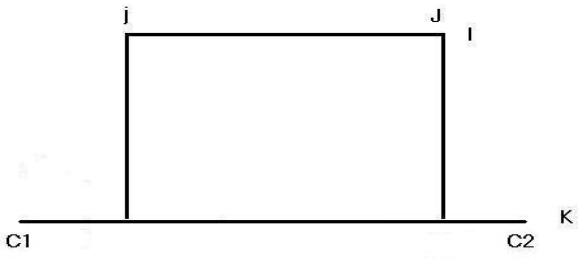
CAS B (c1<j & c2=j) : lorsque C1 est inférieur à j et C2 est égal à j du rectangle courant alors nous ouvrons deux nouveaux rectangles de coordonnées (C1, k, C2, -1) et (j , i, C2, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1, i, C2, k-1) c'est à dire que nous ajoutons ce sommet dans la liste des rectangles fermés.



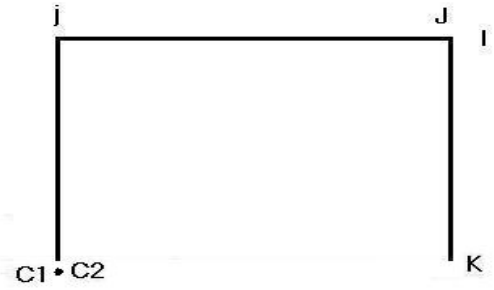
CAS C (c1<j & c2<J) : lorsque C1 est inférieur à j et C2 est compris entre j et J du rectangle courant alors nous ouvrons deux nouveaux rectangles de coordonnées (C1, k, C2, -1) et (j , i, C2, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1, i, C2, k-1) c'est à dire que nous ajoutons ce sommet dans la liste des rectangles fermés.



CAS D (c1<j & c2=J) : lorsque C1 est inférieur à j et C2 est égal à J du rectangle courant alors nous ouvrons un nouveau rectangle de coordonnées (C1, k, C2, -1) dans la liste des rectangles ouverts, de plus le rectangle courant (j, i, J, k-1) est prolongé.

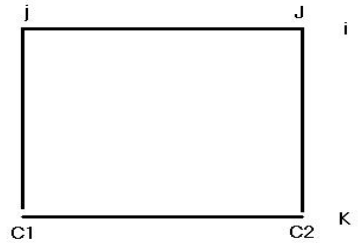


CAS E (c1<j & c2>J) : lorsque C1 est inférieur à j et C2 est supérieur à J du rectangle courant alors nous ouvrons un nouveau rectangle de coordonnées (C1, k, C2, -1) dans la liste des rectangles ouverts, de plus le rectangle courant (j, i, J, k-1) est prolongé.

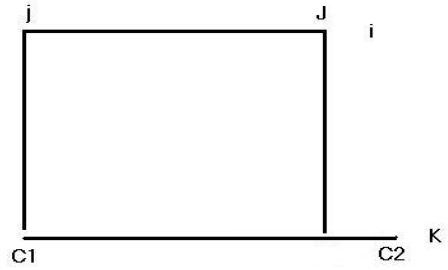


CAS F (c1=j & c2=j) : si C1 et C2 du segment courant sont égaux à j du rectangle courant de la liste des rectangles ouverts et si j est distinct de J, nous ajoutons dans celle-ci un nouveau rectangle de coordonnées (C1,k,C2,-1) et nous fermons ensuite dans la liste des rectangles fermés le rectangle courant (j, i, J, k-1) .

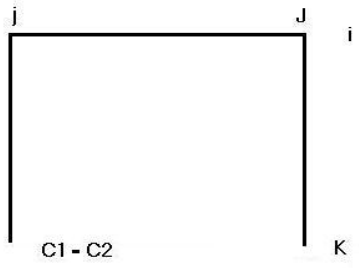
CAS G (c1=j & c2<j) : lorsque C1 est égal à j et C2 est compris entre j et J du rectangle courant alors nous ouvrons un nouveau rectangle de coordonnées (C1, i, C2, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1, i, C2, k-1) c'est à dire que nous ajoutons ce rectangle dans la liste des rectangles fermés.

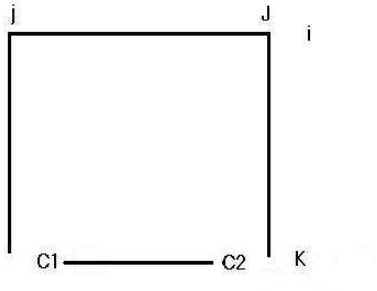


CAS H (c1=j & c2=J) : lorsque C1 est égal à j et C2 est égale à J du rectangle courant de la liste ouverte, nous prolongeons ce rectangle (nous ne faisons rien)

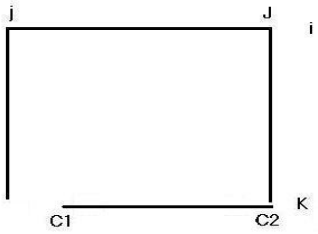


CAS I (c1=j & c2>J) : lorsque C1 est égal à j et C2 est supérieur à J du rectangle courant alors nous ouvrons un nouveau rectangle de coordonnées (C1, k, C2, -1) dans la liste des rectangles ouverts.

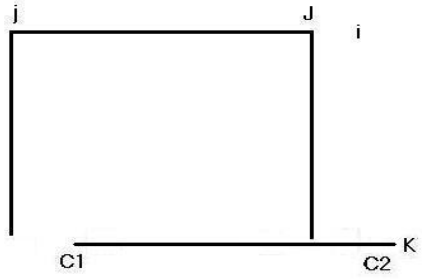




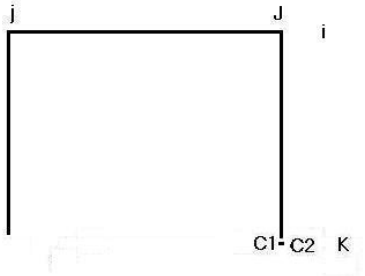
CAS J (c1>j & c2<J) : lorsque C1 et C2 du segment courant sont compris entre j et J du rectangle courant de la liste des rectangles ouverts, nous ajoutons dans celle-ci un nouveau rectangle de coordonnées (C1,k,C2,-1)et nous fermons le rectangle courant de la liste ouverte. Ce cas est le même lorsque C1 et C2 sont égaux.



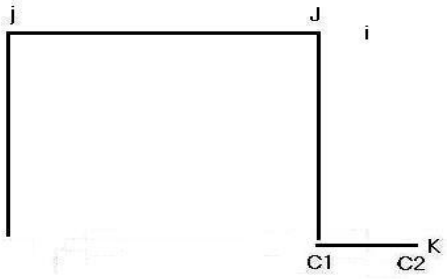
CAS K (c1>j & c2=J): lorsque C1 est compris entre j et J et C2 est égal à J du rectangle courant alors nous ouvre un nouveau rectangle de coordonnées (C1, i, J, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1, i, C2, k-1) c'est à dire que nous ajoutons ce rectangle dans la liste des rectangles fermés.



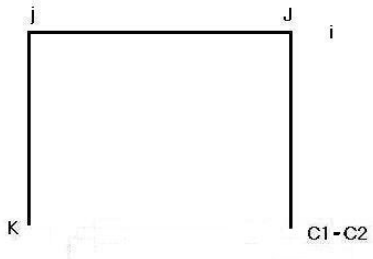
CAS L (c1>j & c2>J) : lorsque C1 est compris entre j et J et C2 est supérieur à J du rectangle courant alors nous ouvrons deux nouveaux rectangles de coordonnées (C1, k, C2, -1) et (C1 , i, J, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1,i,C2, k-1) c'est à dire que nous ajoutons ce rectangle dans la liste des rectangles fermés.

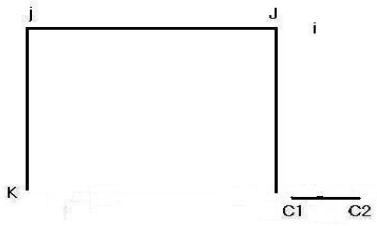


CAS M (c1=J & c2=J) : lorsque C1 et C2 du segment courant sont égaux à J du rectangle courant de la liste des rectangles ouverts, on ajoute dans celle-ci un nouveau rectangle de coordonnées (C1,k,C2,-1). Et nous fermons ensuite dans la liste des rectangles fermés le rectangle courant (j, i, J, k-1) .



CAS N (c1=j & c2>J) : lorsque C1 est égal à J et C2 est supérieur à J du rectangle courant alors nous ouvrons deux nouveaux rectangles de coordonnées (C1, k, C2, -1) et (C1 , i, J, -1) dans la liste des rectangles ouverts, ensuite nous fermons le rectangle courant (C1, i, C2, k-1) c'est à dire que nous ajoutons ce rectangle dans la liste des rectangles fermés.





CAS O (c1>J & c2>J) : lorsque C1 et C2 du segment courant sont supérieurs à J du rectangle courant de la liste des rectangles ouverts, on ajoute dans celle-ci un nouveau rectangle de coordonnées (C1, k, C2, -1). Ce cas est le même lorsque C1 et C2 sont égaux.

D] Problèmes rencontrés

Au cours du développement de l'algorithme, nous avons rencontrés trois problèmes importants:

- Le premier problème rencontré fût de constater que l'algorithme de la thèse était faux, nous avons donc réétudier tous les cas et nous les avons implanté à l'algorithme.
- Pendant l'écriture ou la lecture du fichier, nous avions des problèmes de stockage en données binaires, ces données étaient en fait au format texte brut. Le problème est résolu par l'insertion d'options dans les fonctions de lecture/écriture du fichier. Au début, nous n'avions pas pris en compte car nous pensions que c'était un problème du compilateur.
-

V. Les Algorithmes de démonstration

a. Analyse de la fonction de visualisation

On veut pouvoir visualiser tous les rectangles blancs trouvés à partir des algorithmes d'extraction. Pour cela, on crée une fonction « saveRectRAW » qui enregistre dans un fichier « .RAW » l'image des rectangles optimaux trouvés. On lui passe en paramètre le nom du fichier dans lequel on enregistrera l'image finale, la liste de rectangles, la hauteur et la largeur de l'image, et le nombre de rectangles que l'on veut dessiner.

Analyse

Pour réaliser cette fonction, on se sert des structures de données déjà définies pour les algorithmes d'extraction. Dans la fonction « saveRectRAW », après avoir ouvert le fichier de destination qui sera un fichier binaire de hauteur et de largeur égales à celles passées en

paramètre, je crée une image mémoire dans laquelle les rectangles optimaux seront dessinés. On l'initialise avec des 255, représentant en fait la couleur blanche pour une image vierge. On dessine les contours du rectangle, côtés horizontaux et verticaux avec une couleur grise (en 100) et on remplit ce même rectangle en gris plus foncé (en 50, presque noir). Enfin, on enregistre la nouvelle matrice, dans le fichier « .RAW ». On peut ainsi visualiser la nouvelle image créée, avec les rectangles trouvés, à partir de n'importe quel logiciel de photos, comme Paint Shop ou Image In.

Conception

DEBUT
Ouvrir en écriture le fichier binaire de destination (en ".RAW")
Déclarer un tableau de caractères à deux dimensions
Initialiser le tableau-image en blanc
Se placer au début de la liste de rectangles
Tant que la liste n'est pas vide et que le nombre de rectangles à dessiner n'est pas atteint, faire
    Récupérer le rectangle courant
    Parcourir toutes les lignes de l'image
        Si la ligne de l'image correspond à l'un des côtés horizontaux du rectangle courant
            Parcourir toutes les colonnes de l'image
                Si la colonne de l'image correspond aux colonnes comprises entre les deux côtés verticaux du rectangle courant
                    Dessiner les côtés horizontaux en 100 (gris)
        Sinon
            Si la ligne de l'image correspond aux lignes comprises entre les deux côtés horizontaux du rectangle courant
                Parcourir toutes les colonnes de l'image
                    Si la colonne de l'image correspond à l'un des côtés verticaux du rectangle courant
                        Dessiner les côtés verticaux en 100 (gris)
                    Si la colonne de l'image correspond aux colonnes comprises entre les deux côtés verticaux du rectangle courant

Remplir le rectangle en gris foncé (50)

Passer au rectangle suivant

Augmenter le nombre de rectangles dessinés

Enregistrer le tableau-image dans le fichier de destination

Fermer le fichier

FIN

Problèmes rencontrés

Le problème des couleurs (type de donées):

Un entier prend, en C/C++, deux octets alors qu'un caractère ne prend qu'un seul octet en mémoire. C'est pour cette raison que l'on "cast" les entiers en char sinon le logiciel troquerait les entiers, ce qui au final donnerait une image noire. De plus, il faut savoir que la couleur noire est représentée par 0 et la couleur blanche par 255. Tous les entiers compris entre 0 et 255 correspondent à la couleur grise (plus ou moins foncée).