# Projet d'algorithmique
# Vectorisation d'images

Alexandra Bac

IRM 3

L'objet de ce projet est d'implémenter algorithme de vectorisation d'images, c'est-à-dire, un algorithme permettant de transformer une image noir et blanc en une image vectorielle.

## 1 Approche

L'approche sera basée sur un algorithme d'amincissement permettant d'extraire, à partir de l'image, un squelette. C'est ce squelette qui sera ensuite vectorisé. Donc l'algorithme sera composé des étapes suivantes :

- Calcul d'un squelette de l'image par amincissement (transformée en distance)
- Analyse de ce squelette pour en extraire une vectorisation

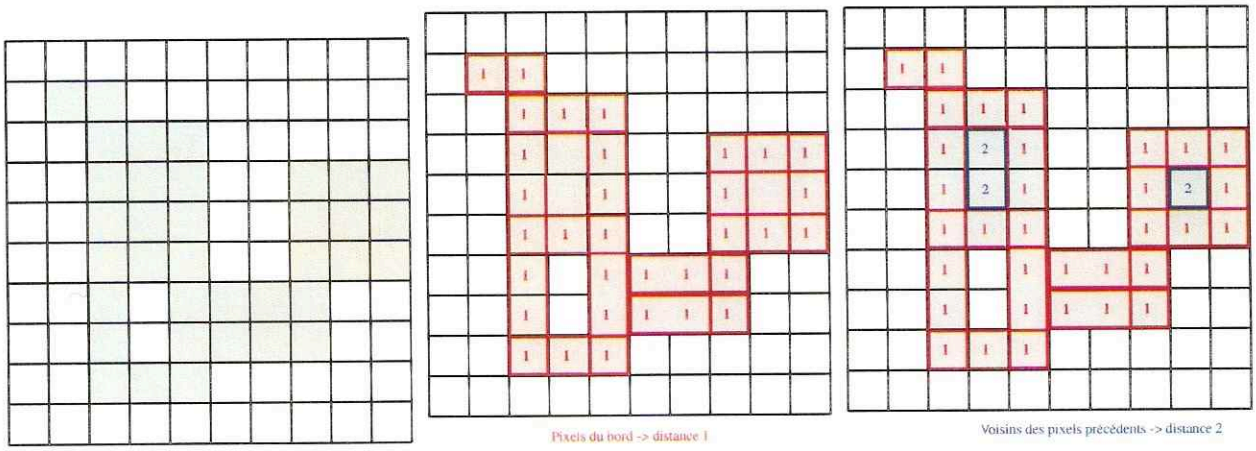
## 2 Amincissement, calcul du squelette

L'amincissement d'une image binaire est basé sur un algorithme très intuitif : on érode l'image en partant des pixels du bord jusqu'à n'obtenir que des lignes de pixels contingus. Ces lignes sont précisement le squelette cherché.

Pour cela, l'algorithme se compose de deux étapes :

- on calcule tout d'abord une transformée en distance de l'image : à chaque pixel on associe sa distance au fond de l'image.
- Puis un squelette sera extrait à partir de cette carte de distance

**La transformée en distance** a pour but de calculer la distance de chaque pixel de l'image au bord :




L'algorithme de calcul de la transformée en distance part de la matrice de l'image (1 dans l'image, 0 hors de celle-ci) comporte deux passes (c'est-à-dire deux traversées de l'image) :

- passe avant : l'image est traversée en priorité sur les lignes (de haut en bas et de gauche à droite).

  On considère pour chaque pixel $p$ les voisins N et O (notés $p_O$ et $p_N$).

  La valeur de chaque pixel est remplacée par le minimum de ces voisins plus 1 :

  $$d(p) = \min(d(p_O), d(p_N)) + 1$$

- passe arrière : l'image est re-traversée de bas en haut et de droite à gauche.

  On considère pour chaque pixel $p$ les voisins S et E (notés $p_S$ et $p_E$).

  Chaque pixel est remplacé par le minimum de lui-même et de ses voisins plus 1 :

  $$d(p) = \min(d(p), d(p_E) + 1, d(p_N) + 1)$$

**La squelettisation** se fait alors par un algorithme similaire à [1]. Le principe est d'attacher des labels aux pixels parmi : *objet*, *fond* et surtout *interne*, *contour*, *multiple* et *squelette*.

- Les pixels de l'image sont marqués *objet* et les autres *fond*.
- Les pixels *objet* de distance 1 sont marqués *contour* et les autres *interne* (donc $d(p) \geq 2$).

L'algorithme s'applique alors itérativement sur les pixels du contour (couche par couche, c'est-à-dire à distance croissante) ; à chaque étape :

- certains sont marqués comme multiples (condition de multiplicité ci-dessous)

- puis :
  - les pixels multiples sont marqués *squelette* et restent dans la classe des pixels *contour*
  - les autres sont supprimés (passé dans le *fond*)
  - puis la couche suivante de pixel est marqués comme *contour* et la boucle est répétée

Un pixel est marqué *multiple* si au moins une des conditions suivante s'applique :

- pour les paires de voisins N-S et E-O, les pixels ne sont pas l'un dans le *fond* et l'autre *interne*,
- chaque triplet de voisins $\{N{-}NE{-}E, E{-}SE{-}S, S{-}SO{-}O, O{-}NO{-}N\}$, est tel que le pixel diagonal appartient au *contour* alors que les deux autres sont dans le *fond*.

L'algorithme est itéré contour après contour jusqu'à ce que tous les pixels du contour courant soient finalement marqués *multiples*. L'ensemble des points multiples constitue alors le squelette.

# 3 Vectorisation

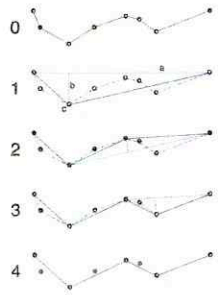
A partir du squelette, l'étape suivante consiste à extraire un ensemble de vecteurs (ou lignes polygonales) approximant le squelette. Cette polygonalisation du squelette sera effectuée grâce à l'algorithme de Douglas-Peucker. Cet algorithme simplifie des lignes polygonales (suite de pixels voisins) et dépend d'un seuil $\varepsilon$ fixant sa précision.

L'algorithme travaille de manière récursive par la méthode "diviser pour régner".

À l'initialisation on sélectionne le premier et le dernier nœud de la ligne à simplifier. Ce sont les bornes.

À chaque étape on parcourt tous les nœuds entre les bornes et on sélectionne le nœud le plus éloigné (en terme de distance du noeud au segment) du segment formé par les bornes :

- s'il n'y a aucun nœud entre les bornes l'algorithme se termine,
- si cette distance est inférieure au seuil $\varepsilon$ on supprime tous les nœuds entre les bornes,
- si elle est supérieure la polyligne n'est pas directement simplifiable. On appelle de manière récursive l'algorithme sur deux sous-parties de la polyligne : de la première borne au nœud distant, et du nœud distant à la borne finale.




Pour pouvoir appliquer cet algorithme, la question est donc d'isoler les polylignes à simplifier : les vecteurs. Pour cela, le squelette est tout d'abord segmenté en lignes polygonales (éventuellement fermées) distinctes. Les vecteurs sont identifiés par un label et chacun comporte deux extrémités. Les vecteurs seront calculés par simple parcours du squelette :

- Partant d'un point quelconque (et initialisation un vecteur sur ce point), le squelette est parcouru de proche en proche et le vecteur étendu jusqu'à ce que :
  - soit on atteigne une extrémité
  - soit on atteigne un point de confluence
  - soit le vecteur revienne sur son autre extrémité
- Dans tous ces cas, on parcourt et crée alors le vecteur suivant soit à partir du point de confluence, soit à partir d'un point quelconque non encore marqué.

# 4 Format d'entrée

Les images seront lues en format BMP, qui est un format bitmap simple. Pour cela, on utilisera la bibliothèque développée par Thomas Bore :

`http://bores.fr/blog/2011/07/langage-c-jouons-avec-les-fichiers-bmp-la-suite/`

Je vous fournirai les sources.

# 5 Format de sortie

Afin de pouvoir facilement visualiser les vectorisations, la sortie de fera en Latex (qui une fois compilé produira un pdf vectoriel).

Vous créerez un fichier latex, i.e. d'extension `.tex` de la forme suivante :

```
\documentclass{article}

\usepackage{tikz}
```

```
\begin{document}

\begin{tikzpicture}
...
\end{tikzpicture}

\end{document}
```

Pour dessiner un segment entre les points `(i1, j1)` et `(i2, j2)`, dans la `tikzpicture` :

```
\draw (i1,j1) -- (i2,j2)
```

La visualisation du résultat se fait en compilant le fichier latex (en ligne de commande) : `latex nom_du_fichier`

## References

[1] Arcelli C., Sanniti di Baja G., A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform, IEEE Trans. Pattern Analysis and Machine Intelligence, 11(4): 411-414, 1989.