

# Aquisition et analyse d'image

Hadrien Croubois

Nicolas Lourdeau

Partie 1

## Philosophie d'utilisation

Dans le cadre de ce projet, les outils développés ont toujours été prévu pour être réutilisés. Ainsi le code se décompose tout naturellement en deux parties, d'un coté une librairie facilement compilable au format `.so` ou `.dll` fournit avec les fichiers en-tête correspondant et d'autre part un programme simple appelant les fonctionnalités de la bibliothèque.

Afin de donner différents exemples d'utilisation de la bibliothèque *lenactions*, deux applications sont fournies et donnent ainsi différents exemples d'utilisation plus ou moins haut niveaux des fonctionnalités de la bibliothèque.

- **Lenaction** : programme simple, effectuant des appels à la bibliothèque à partir des arguments codés en dur dans le programme.
- **LenaSH** : un shell minimaliste interprétant des commandes écrites dans un langage ad-hoc et proposant ainsi une interface haut niveau pour l'utilisateur.

Partie 2

## Outils utilisés

Afin d'optimiser la portabilité de la bibliothèque, cette dernière ne dépend d'aucun autre outil. Ecrite en C/C++ elle gère toutes les étapes du traitement d'images, du chargement de fichier au calcul de contours, en passant par la transition dynamique entre les espaces de couleur RGB et HSV.

Le programme **LenaSH** utilise les outils FLEX et BISON pour la reconnaissance du langage ad-hoc développé parallèlement à la librairie.

Afin de simplifier l'étape de compilation des différents composants du programme sur différentes architectures, nous utilisons l'outil CMAKE ainsi qu'un script `./generator`.

Partie 3

## Fonctionnalités

De nombreux algorithmes sont actuellement déployés dans la bibliothèque à différents niveaux :

- Pixel :
  - Conversion d'espace de couleur
  - Opérateurs de fusion (quadratique, angle)
- Image :
  - Chargement/Sauvegarde au format `.ppm` / `.pgm`
  - Calcul de seuil
  - Composition avec un filtre
  - Assemblage de deux images
  - Seuillage (local, global, histerisis)
  - Affinage de contour
  - Calcul de contour fermés

La mise en place de filtre de convolution standard permet par ailleurs de calculer simplement les contours via les filtres de Prewitt, Sobel et Kirsch ainsi que d'appliquer un filtre moyennneur gaussien pour lisser l'image.

Partie 4

## Algorithme

4.1

### seuillage par histerisis

Cette opération de seuillage revient à calculer des composantes connexes pour le critère de luminosité ( $> low$ ) dans l'image. Pour cela on utilise une structure de *union find* qui garantit un résultat rapide  $\mathcal{O}(n.Ack^{-1}(n))$ .

L'ajout d'un drapeau au niveau des composantes connexes permet de marquer les composantes dont un des éléments vérifie le critère de luminosité ( $> high$ ).

On ne garde ensuite que les pixels appartenant à une composante connexe marquée.

4.2

### Affinage des contours

Pour cette opération, on se base sur une image de contour obtenue à partir de l'opérateur de mélange `angle` appliqué à deux gradients.

L'angle décrivant localement le contour (azimut du gradient) est discrétisé et permet de parcourir localement la largeur du contour. En ne gardant que le pixel au centre de ce contour on arrive à garder un contour fidèle, peu bruité et limitant les trous.

4.3

### Fermeture des contours

Pour cette opération, nous avons développé un algorithme à vagues, proche de ceux utilisés en système distribué pour la communication sur des grilles de processeurs.

Ici chaque pixel est une entité pouvant être dans l'un des quatre états suivants :

- *Vide*
- *Champ*
- *Contour*
- *Ancre*

Au début de l'algorithme, on part d'un contour affiné dont les pixels sont dans l'état *Contour* tandis que le fond est dans l'état *Vide*.

La première passe se charge de détecter les *Ancre*s parmi les pixels du *Contour*. Pour cela on détecte tout ceux qui ont soit aucun voisin (ancres d'adjacence 2) soit ceux qui sont en bord de contour (un unique voisin ou deux voisins collés) qui sont des ancres d'adjacence 1.

Les ancres sont les points d'intérêt qu'il s'agit de relier. Leur adjacence correspond au nombre de liaison à former pour faire partie d'un contour fermé.

A partir de là, on applique un algorithme de diffusion pour répandre un champ autour des ancres. Les pixels passent ainsi de l'état *Vide* à l'état *Champ*.

La jonction de champs provoque la liaison des deux ancres et la diminution de leur adjacence.

Si un champ rencontre un pixel d'un contour et que celui-ci n'est pas dans la composante connexe de l'ancre associé au champ, il s'y accroche, faisant par la même occasion diminuer l'adjacence de l'ancre dont il provient.

Pour cette diffusion on utilise une file de priorité implémentée à partir d'un tas et une distance basée sur BLUE (best linear unbiased evaluation) qui prend aussi en compte les variations successives du gradient le long du contour à retrouver.

Cet algorithme donne de bons résultats dès lors que la taille des trous est inférieure à la taille des composantes connexes à fermer.

Voir la partie A (page 3).

Partie 5

### Conclusion

Après de longues séances de recherche, nous sommes satisfaits de la forme que prend notre bibliothèque. De nombreux outils précédemment développés dans le cadre de `LibImAMXX` (bibliothèque graphique développée par Hadrien Croubois en M1 et qui a servi de support) sont actuellement en cours d'ajout. Parmi ces outils on compte notamment l'affichage des histogrammes avec `gnuplot`, la normalisation des espaces de couleurs ainsi que le tatouage.

Pour ce qui est des algorithmes développés, nous sommes relativement contents des résultats renvoyés par l'algorithme d'affinage.

L'utilisation de champs dans un algorithme à vagues nous semble une piste intéressante pour la fermeture des contours. Beaucoup de travail reste cependant à faire, aussi bien pour ce qui est du choix de la fonction de distance que pour les heuristiques de création des liens d'ancrage. Nous espérons pouvoir continuer à améliorer notre algorithme afin de le rendre plus efficace, notamment en s'adaptant dynamiquement aux propriétés de l'image.

Le code du programme est disponible sous licence GNU GPL v3 sur GitHub : <https://github.com/Amxx/Lenactions>

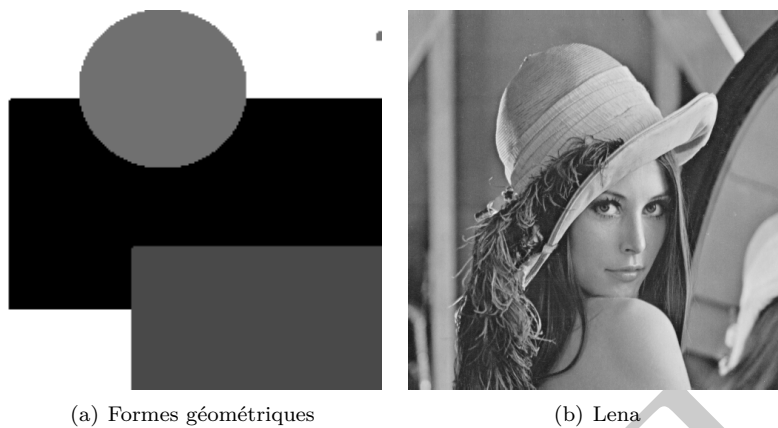


FIGURE 1 – Images utilisées lors des tests

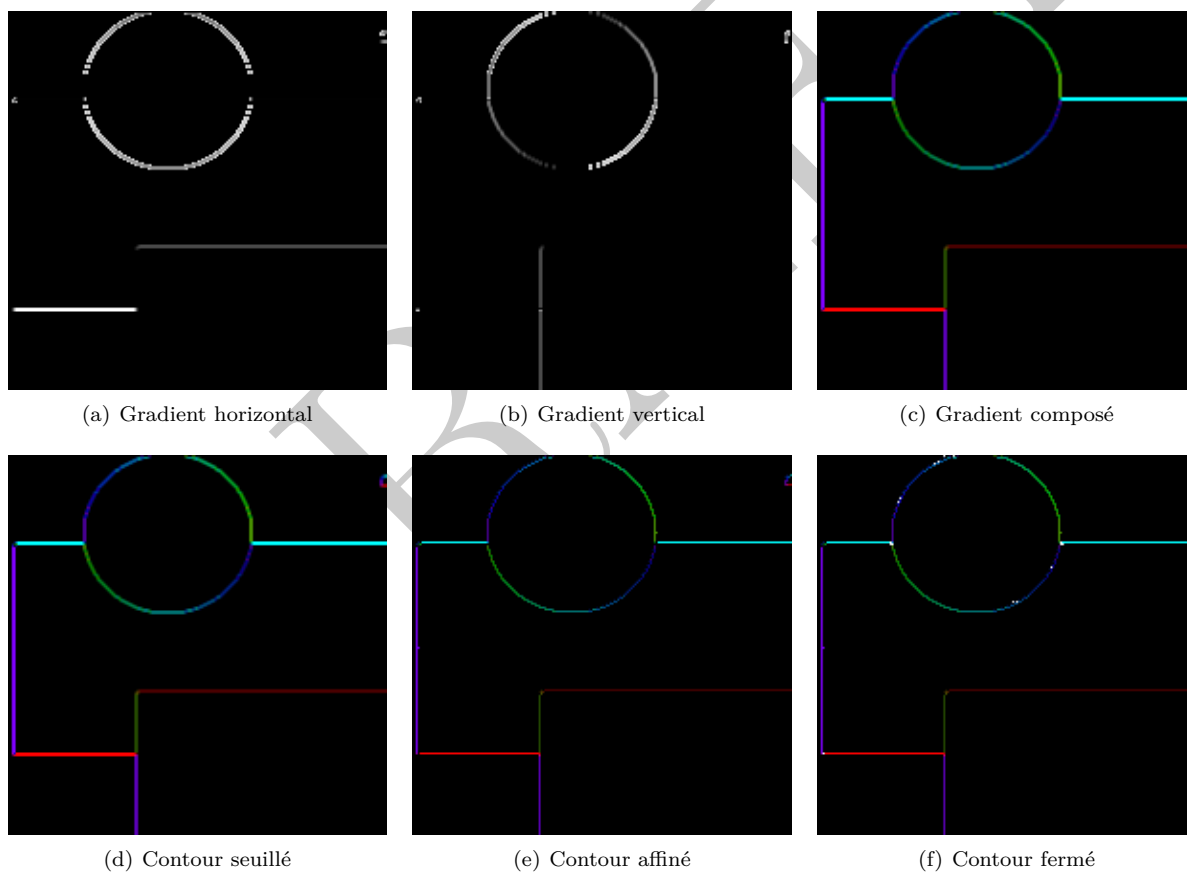


FIGURE 2 – Résultats intermédiaires pour des formes simples

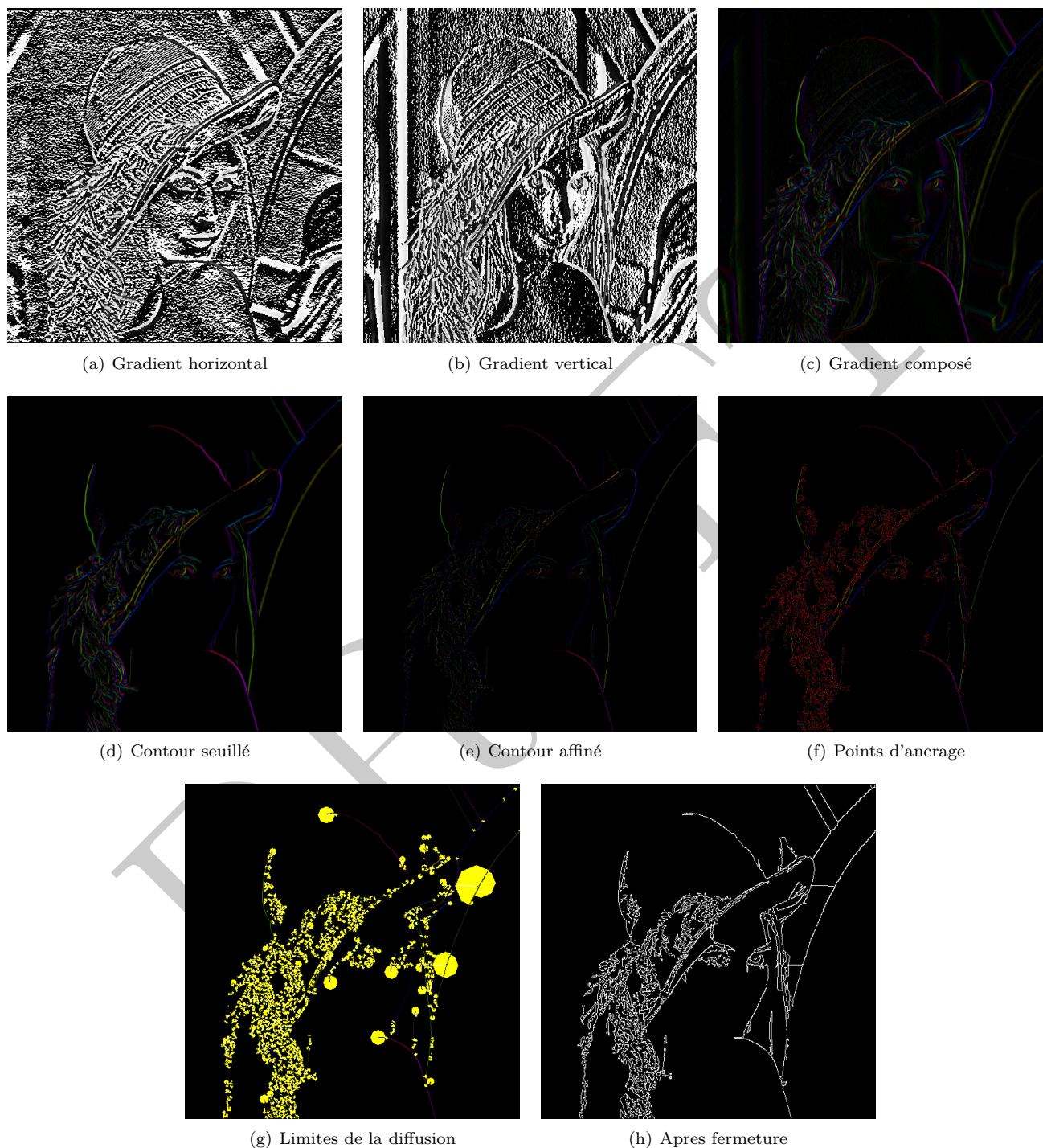


FIGURE 3 – Résultats intermédiaires pour Lena