Arbre des composantes : décomposition d'une image par algorithme Union-Find

BRESNU Arthur, HENON-JUST Guillaume

École des Ponts ParisTech

Gaspard Monge Laboratoire d'informatique

Introduction

Dans le cadre du traîtement d'images, des outils détectant les formes et frontières dans une image peuvent se révéler essentiels. On peut alors s'intéresser aux parties connexes de pixels dans cet image selon leur niveau de gris. Pour stocker l'image en fonction de ces parties connexes de manière efficace, on utilise une structure de donnée en arbre : l'arbre des composantes. Pour calculer cet arbre de manière efficace, nous avons implémenté l'algorithme Union-Find décrit dans [1].

Nous nous sommes ensuite intéressés aux applications d'un tel stockage de l'image.

Abres des composantes

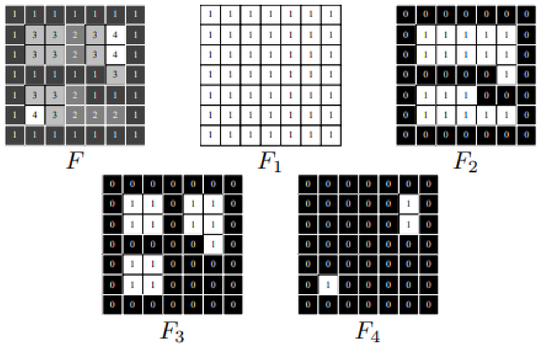


Figure 8: Image de base et ses niveaux de gris

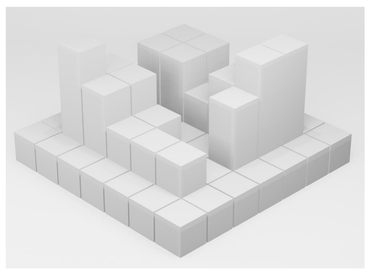


Figure 9: Visualisation 3D des niveaux de gris (altitude)

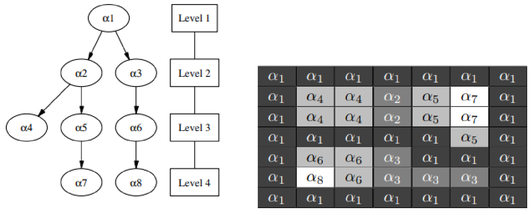


Figure 10: Arbre des composantes de cette image

References

[1] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *IEEE Transactions on Image Processing*, 15(11):3531–3539, 2006.

Algorithme Union-Find

L'algorithme consiste à créer un arbre pour chaque pixel de l'image, et par altitude décroissante on vient fusionner les arbres quand nécessaire jusqu'à n'avoir plus qu'un arbre.

Algorithm 1 Union-Find

Input: $image$
Output: $tree$
function UnionFind($image$)
  forall $pix\_p \in image$ **do**
    $createPartialTree(pix\_p)$
  $P \leftarrow sortPixel(image)$
  forall $lvl \in [\![0, 255]\!]$ **do**
    forall $pix\_p \in P[lvl]$ **do**
      forall $pix\_q \in Neighborhood[pix\_p]$ ***and*** $lvl \leq level[pix\_q]$ **do**
        if $Nodes[pix\_p] \neq Nodes[pix\_q]$ **then**
          $mergeNodes(Nodes[pix\_p], Nodes[pix\_q])$
  return $tree$

Implémentation en C++

Classe **Node :**

- **level** : niveau de gris
- **children** : enfants du noeuds
- **pixels** : pixels du noeuds
- **metric** : mesure de l'aire, hauteur et volume du noeud

Pour atteindre une complexité quasi-linéaire : Chaque noeud a un élément canonique représentant l'arbre partiel auquel il appartient. Chaque pixel a un élément canonique représentant le noeud auquel il appartient. On met à jour au fur et à mesure de l'algorithme ces éléments canoniques.
⇒ Manipulation de tableaux uniquement.

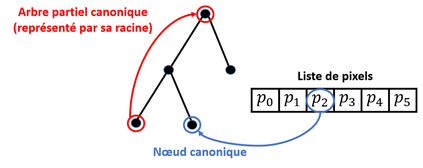
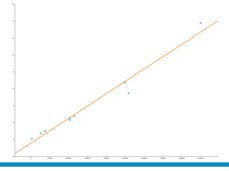


Figure 11: Schéma des éléments canoniques

Performances

Le papier [1] annonce une complexité quasi-linéaire pour l'algorithme Union-Find. Nous avons donc décidé de vérifier ce résultat sur plusieurs images de taille différentes à l'aide d'une régression linéaire.

$R = 0.9848$ avec 10 points.
La complexité semble donc bien quasi-linéaire.



Filtrage des granularités de l'image

Filtrage de l'image en lissant ses imperfections et détails : détection des parties connexes de mesure plus faible qu'un seuil fixe. Ajout d'un champs **area** aux noeuds: mesure le nombre de pixels du noeud et de ses enfants : surface du noeud.

Figure 1: Image originale

Figure 2: Seuil : 1000

Figure 3: Seuil : 10000

Figure 4: Seuil : 500000

Implémentation : ajout du champs **is_considered = False** si l'area du noeud est plus faible que le seuil fixé. On n'affiche alors que les noeuds "considérés" et on lisse les autres.

Visualisation en temps réel des parties connexes

Implémentation adaptée pour obtenir les deux arbres des composantes, croissant et décroissant. Affichage de la partie connexe correspondant au clic :

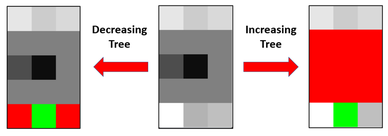


Figure 5: Affichage du noeud sélectionné et de ses enfants

Détection de formes dans une image :

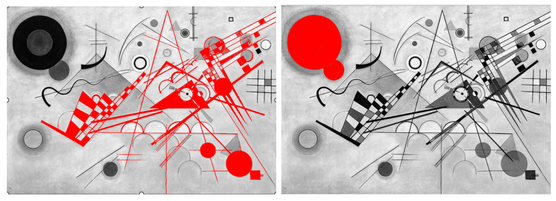


Figure 6: Affichage des parties connexes d'images plus complexes

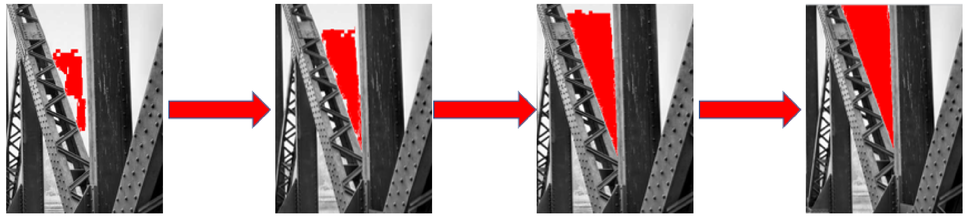


Figure 7: Affichage des parties connexes de l'image en temps réel selon les mouvements de la souris

Conclusion

Nous avons réussi à implémenter l'algorithme Union-Find et avons pu voir quelques-unes de ses applications. Nous avons ainsi pu découvrir des méthodes algorithmiques que nous ne connaissions pas (gestion d'arbre dans des tableaux, représentation d'une partie connexe par élément canonique, ...
Malgré tout, nous avons essayé d'implémenter la méthode **keep_N_lobes** sans succès car nous avons manqué de temps pour aboutir. De plus, malgré nos optimisations, l'affichage en temps réel des parties connexes est satisfaisant mais pas à la hauteur de la fluidité espérée.