

HAI809I — Projet Image et Compression

Parcours Imagine 2024-2025

Mathis Duban (mathis.duban@etu.umontpellier.fr)
Paul Deligne (paul.deligne@etu.umontpellier.fr)

March 2, 2025

Compression basée super-pixels

Compte rendu du 2 mars 2025

1 Travail déposé dans le dépôt GitHub

Durant cette semaine de travail, de nouveaux éléments ont été déposés dans différents dossiers :

- Le Dossier **Articles** contient un article sur la méthode SLIC annoté afin de relever certains détails pour la bonne compréhension des enjeux et du déroulement de l'algorithme.
- Le dossier **Image** contient les images d'entrées et de sorties utilisées pour l'algorithme.
- Le dossier **Liens** contient un fichier texte regroupant plusieurs liens utiles : descriptions de méthodes alternatives et explication de pseudo codes.
- Le dossier **Notes** contient des notes diverses et variées afin de ne pas oublier des éléments au fur et à mesure des semaines comme des pseudos code par exemple
- Le dossier **Code** contient tout le code (fichiers .cpp/.h)

Voici l'adresse de notre repository GitHub contenant tous les éléments actuels du projet:

<https://github.com/Akkuun/Super-Pixel-Project>

2 Travail effectué cette semaine

Toute cette semaine a été dédiée à l'élaboration d'une base solide de code pour le projet et au commencement du développement de la méthode SLICC.

2.1 Choix de la technique SLICC

Tout d'abord, le choix de l'algorithme s'est porté sur la méthode **SLICC**, qui est une extension de la technique SLIC. En effet, SLICC ajoute une étape de vérification des composantes connexes, permettant d'éliminer les super-pixels de trop petite taille. Cela évite la présence de super-pixels parasites en les fusionnant avec des groupes plus massifs. L'objectif final étant de comparer les résultats issus de deux techniques distinctes, il était pertinent de choisir d'abord une méthode "optimisée" comme point de référence, avant de la confronter à une autre approche.

Note : Une autre technique n'utilisant pas de paramètre particulier à été trouvé nommée SLICO, elle permet d'avoir un résultat similaire sans à avoir à varier les paramètres de la méthode SLICC, nous ferons plus tard (si possible) des comparatifs avec cette dernière.

2.2 Crédit d'une base de code stable et propre

Une attention particulière a été portée à la réalisation de la bibliothèque de code de traitement d'image que nous allons utiliser tout au long de ce projet. Dans cette base de code repensée, cela nous permet d'appliquer de manière séparée et distincte des traitements sur des images. Dans cette bibliothèque, on y retrouve :

- La gestion des objets **Image** pouvant être distingué en 3 types de formats : PPM(couleurs), PGM(niveaux de gris), LAB (cf Partie 2.3)
- La gestion mémoire de la création et destruction des objets d'images
- Lecture et écriture d'image ergonomique
- Application de traitement d'image facilement adaptable
- Une création d'une documentation dynamique complète du code (qui pourra être éditée via des outils comme Oxygen potentiellement).

Tout le travail concerné peut être retrouvé dans le dossier [/Code](#) sur le GitHub.

2.3 Avancement par rapport à l'algorithme SLICC

Dans cette partie, nous allons discuter de l'avancement qui à été fait vis-à-vis de SLICC.

Après de nombreuses recherches personnelles, voici un pseudo code écrit visant à réaliser SLICC (retrouvable dans [/Notes](#):

```
// PHASE 1 : Initialisation
//1.1 Convertir l image RGB en CIELab.
//1.2 D finir le nombre de superpixels souhait et calculer le pas d
//1.3 Placer les centres de clusters Ck sur une grille r guli re (av
//1.4 Initialiser la matrice des labels L(x, y) -1 et la matrice d
D(x, y) INF
// PHASE 2 : Assignation des pixels aux clusters
    //2.1 Pour chaque centre de cluster \(\mathbf{C}_k\):
//2.2 Parcourir les pixels dans une fen tre locale de taille \(\mathbf{2S}\times\mathbf{2S}\)
//2.3 Pour chaque pixel \(\mathbf{P}(x, y)\) dans rcette gion :
    // Calculer la **distance couleur** \(\mathbf{d}_{\text{lab}} = ||\mathbf{C}_k^{\text{lab}} - \mathbf{P}^{\text{lab}}||\)
    // Calculer la **distance spatiale** \(\mathbf{d}_{\text{xy}} = ||\mathbf{C}_k^{\text{xy}} - \mathbf{P}^{\text{xy}}||\)
    // Calculer la distance totale :
    // \mathbf{D} = \mathbf{d}_{\text{lab}} + \frac{m}{S} \cdot \mathbf{d}_{\text{xy}}
    // Si \(\mathbf{D} < \mathbf{D}(x, y)\), mettre jour \(\mathbf{D}(x, y)\) et assigner \(\mathbf{L}(x, y)\) à \(\mathbf{C}_k\)

// PHASE 3 : Mise jour des centres des superpixels
//3.1 Pour chaque cluster \(\mathbf{C}_k\):
    // Calculer le **nouveau centre** comme la moyenne des pixels lui appartenant
//3.2 Repeter **PHASE 2 et 3** jusqu' convergence (\(\mathbf{C}_k < \text{seuil}\))

// PHASE 4 : Correction de la connectivit (SLICC sp cifique)
//4.1 Parcourir l'image pour detecter les superpixels non connexes :
// Effectuer un **flood fill** pour identifier les **composantes connexes**
    // Si une composante est **trop petite**, l assigner au superpixel
//4.2 Mettre jour les labels \(\mathbf{L}(x, y)\) apr s fusion des petits groupes
```

Actuellement, voici ce qui a été réalisé :

- Conversion de l'image RGB en CIELAB

- PHASE 2

- PHASE 3

La phase 4 est en cours de développement.

Pour préciser un petit plus le travail autour de CIELAB, l'algorithme SLICC nécessite de travailler sur une image en format CIELAB.

Contrairement au format RGB, le format CIELAB ou LAB permet de représenter une image dans l'espace LAB. Cet espace est perceptuellement uniforme, c'est-à-dire que les différences de couleurs perçues par l'œil humain correspondent mieux aux distances dans cet espace que dans d'autres espaces comme RGB. Nous avons les composantes :

- L (Luminance) → Intensité lumineuse (0 = noir, 100 = blanc).
- a → Représente l'axe vert/rouge (-128 = vert pur, +127 = rouge pur).
- b → Représente l'axe bleu/jaune (-128 = bleu pur, +127 = jaune pur).

La transformation de RGB à LAB peut se faire en suivant ces différentes instructions :

```
// PHASE 1 : Conversion RGB      XYZ
1.1 Normaliser chaque canal RGB dans [0,1] :
    R' = R / 255
    G' = G / 255
    B' = B / 255
1.2 Appliquer la correction gamma inverse (sRGB -> Linéaire) :
    Si (R' <= 0.04045) alors R' = R' / 12.92
    Sinon R' = ((R' + 0.055) / 1.055) ^ 2.4
    (Même chose pour G' et B')
1.3 Calculer les valeurs XYZ avec la matrice de transformation :
    X = 0.4124564 * R' + 0.3575761 * G' + 0.1804375 * B'
    Y = 0.2126729 * R' + 0.7151522 * G' + 0.0721750 * B'
    Z = 0.0193339 * R' + 0.1191920 * G' + 0.9503041 * B'

// PHASE 2 : Conversion XYZ -> CIELab
2.1 Normaliser les valeurs par les **valeurs de référence** (illumination)
    X = X / 95.047
    Y = Y / 100.000
    Z = Z / 108.883
```

```
2.2 Appliquer la transformation de **fonction f(t)** :  
    f(t) = t^(1/3) si t > 0.008856  
          (7.787 * t) + (16 / 116) sinon  
2.3 Calculer les composantes Lab :  
    L = 116 * f(Y) - 16  
    a = 500 * (f(X) - f(Y))  
    b = 200 * (f(Y) - f(Z))
```

```
// PHASE 3 : Stocker les valeurs (L dans [0,100], a et b dans [-128,128])
```

Pour notre projet, on souhaite voir les effets de compression via cette technique, j'ai donc utilisé une image HD assez large afin de travailler avec, voici ce que nous avons obtenu en séparant chaque composante dans une image à part :



Figure 1: Image de base en format RGB (3456 x5184)



Figure 2: Composante L issue de la conversion dans l'espace LAB



Figure 3: Composante A issue de la conversion dans l'espace LAB



Figure 4: Composante B issue de la conversion dans l'espace LAB

Il ne nous reste plus que la PHASE 4 à réaliser et essayer ensuite de faire tourner nos algorithmes SLICC.

3 Travail à venir

Pour le travail à venir, nous allons essayer de faire fonctionner la phase 4 de notre feuille de route pour SLICC et tenter de créer nos premières images super-pixelisés.

4 Références

References

- [1] Bibilothèque d'image Unsplash,
<https://unsplash.com/fr/s/photos/reptile>
- [2] jflalonde,
<http://vision.gel.ulaval.ca/~jflalonde/cours/4105/h17/tps/results/projet/111063028/index.html> *Segmentation d'images en superpixels via SLIC*
- [3] jflalonde, epfl
<https://www.epfl.ch/labs/ivrl/research/slic-superpixels/#SLICO> *SLIC Superpixels*