HAI809I — Projet Image et Compression
Parcours Imagine 2024-2025

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

March 9, 2025

Compression basée super-pixels

Compte rendu du 9 mars 2025

1 Travail déposé dans le dépot GitHub

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

- Le dossier **Image** contient les images d'entrées et de sorties utilisées pour l'algorithme.
- 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 terminer l'implémentation de la méthode SLICC.

2.1 Travail effectué auparavant

à la fin de la semaine dernière, nous avions créé une bibliothèque de code de traitement d'image que nous utiliserions tout au long du projet. Nous avions également implémenté une méthode pour transformer nos images du format RGB au format CIELab. Enfin, nous avions commencé à implémenter l'algorithme SLICC.

2.2 Fin de l'algorithme SLICC

Dans cette partie, nous allons discuter de la fin de l'implémentation de SLICC.

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 c
//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 c
D(x, y)    INF
// PHASE 2 : Assignation des pixels aux clusters
    //2.1 Pour chaque centre de cluster \( C_k \) :
//2.2 Parcourir les pixels dans une fen tre locale de taille \( 2S \t
//2.3 Pour chaque pixel \( P(x, y) \) dans cette  gion  :
//- Calculer la **distance couleur** \( d_{lab} = || C_k^{lab} - P^{la
//- Calculer la **distance spatiale** \( d_{xy} = || C_k^{xy} - P^{xy}
//- Calculer la distance totale :
//D = d_{lab} + \frac{m}{S} \cdot d_{xy}
//- Si \( D < D(x, y) \), mettre    jour \( D(x, y) \) et assigner \(

// PHASE 3 : Mise    jour des centres des superpixels
```

```
//3.1 Pour chaque cluster \( C_k \) :
//- Calculer le **nouveau centre** comme la moyenne des pixels lui app
//3.2 R p ter **PHASE 2 et 3** jusqu'  convergence ( Ck  < seuil).
// PHASE 4 : Correction de la connectivit  (SLICC sp cifique)
//4.1 Parcourir l'image pour d tecter les superpixels non connexes :
//Effectuer un **flood fill** pour identifier les **composantes connex
//- Si une composante est **trop petite**,  l assigner  au superpixel
//4.2 Mettre    jour les labels \( L(x, y) \) apr s fusion des petits
```

Voici ce qui a été réalisé la semaine dernière pour l'implémentation de SLICC:

- Conversion de l'image RGB en CIELAB
- PHASE 2
- PHASE 3

Cette semaine nous avons fait :

- Régler le problème observée de la convergence qu'on n'atteignait pas pour la partie 3.2
- Fin du développement de la PHASE 4
- Conversion de l'image CIELAB en RGB

La transformation de LAB à RGB peut se faire en suivant ces différentes instructions qui sont pour simplifier les instructions et opérations à effectuer dans l'ordre inverse par rapport à la conversion RGB à LAB développées dans le précédent compte rendu:

```
// PHASE 1 : Conversion LAB -> XYZ
1.1 Calculez les valeurs t de la fonction inverse de conversion
pour chaque canal Lab :
    ty = (L + 16.0) / 116.0;
    tx = ty + (A / 500.0);
    tz = ty - (B / 200.0);
1.2 Appliquer la fonction inverse de conversion au format XYZ :
    Si tx > 0.206893 alors X = 95.047 * tx^3
    Sinon X = 95.047 * (tx - 16.0 / 116.0) / 7.787)
```

```
    Pour Y et Z on fait la meme chose sauf qu'on change 95.047
    par 100 pour Y et 108.883 pour Z

// PHASE 2 : Conversion XYZ -> RGB
2.1 Calculer les valeurs RGB avec la matrice de transformation inverse
    R =  3.2406 * (X/100.0) - 1.5372 * (Y/100.0) - 0.4986 * (Z/100.0)
    G = -0.9689 * (X/100.0) + 1.8758 * (Y/100.0) + 0.0415 * (Z/100.0)
    B =  0.0557 * (X/100.0) - 0.2040 * (Y/100.0) + 1.0570 * (Z/100.0)

2.2 Appliquer la correction gamma (Lineaire -> Lineaire) :
    Si (R <= 0.0031308) alors R = R * 12.92
    Sinon R = 1.055 * R ^ (1 / 2.4) - 0.055
    (Meme chose pour G et B)

// PHASE 3 : Stocker les valeurs R, G et B dans [0,255]
```

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 (606 x 622)

Figure 2: Image de base après une conversion au format LAB puis une reconversion au format RGB

Figure 3: Composante L à l'issue de l'algorithme SLICC

Figure 4: Composante A à l'issue de l'algorithme SLICC

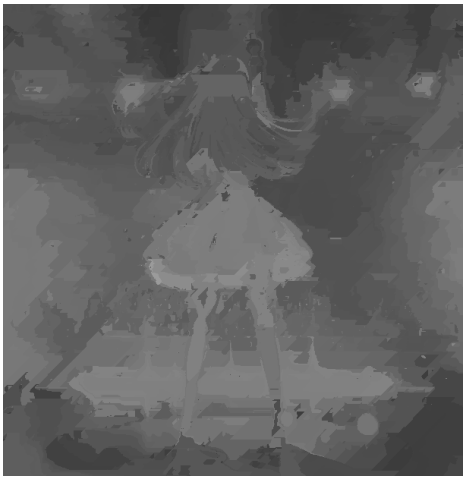

Figure 5: Composante B à l'issue de l'algorithme SLICC

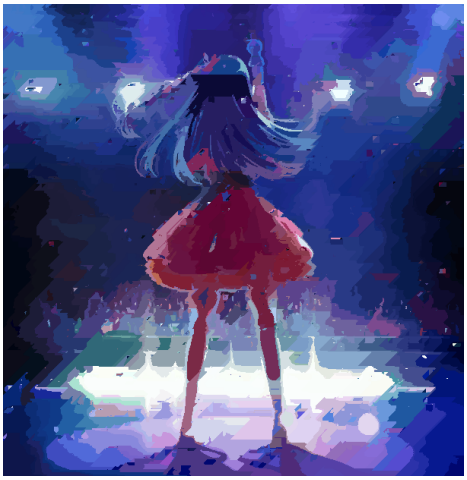

Figure 6: Image de Base à l'issue de l'algorithme SLICC

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 d'implémenter d'autres méthodes, ainsi qu'essayer de résoudre un problème avec l'image présentée la semaine dernière. En effet, cette dernière provoque une erreur de segmentation et on remarque qu'à partir d'une certaine taille de l'image, cela en provoque une.

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*