Compte rendu 4

Traitement harmonique des images

Partie 7 : Changement de technologie

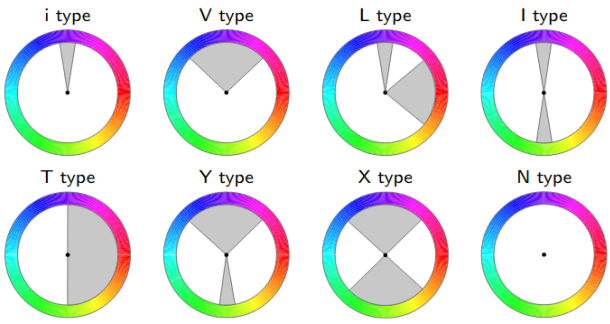
Jusqu'ici, nous utilisions du python pour Tan et al, et du C++ pour Cohen-or.

Le C++, bien qu'ayant l'avantage de la vitesse de traitement, nous ralentissait énormément dans le développement, car nous devions créer la librairie de traitement d'images en même temps. Nous avons décidé de tout traduire en python.

La transition ne fut pas trop difficile, et le développement en est accéléré. D'autre part, Cela nous donne la possibilité de rendre notre implémentation de Cohen-or compatible avec le backend de l'interface utilisateur créée par Donovann, ce qui nous permettra de proposer et de pouvoir comparer les deux méthodes au travers de l'interface.

Partie 8 : Cohen-or et al, distance à une harmonie

L'algorithme de ce papier fonctionne en deux étapes. On part d'une liste de templates harmoniques, qui représentent des zones dans l'espaces des teintes (donc dans $\frac{\mathbb{R}}{360\mathbb{Z}}$) telles que, si les teintes d'une image sont dedans, elles sont considérées comme harmoniques. Voici la liste des templates :



La première étape consiste à, étant donné une image, trouver le « meilleur template » pour cette image, en d'autres termes, le template le plus proche (à une constante près) de la répartition des teintes de l'image.

Pour cela, il nous faut d'abord créer quelques fonctions permettant de manipuler les valeurs de l'espace quotient des teintes :

```python
def angle_distance(a, b):
    a = a % 360
    b = b % 360
    return min(
        abs(a - b),
        abs(a + 360 - b)
    )

def between(v, a, b):
    v = v % 360
    a = a % 360
    b = b % 360
    if b > a: return a <= v <= b

    return (v >= a) or (v <= b)
```

On utilise la classe utilitaire suivante pour manipuler les templates. **HarmonicTemplate::sectors** contient des tuples (*angle d'ouverture*, *angle de décalage*). Par exemple, le template « T » est défini par $[(180, 90)]$

```python
class HarmonicTemplate:
    sectors = []

    def __init__(self, sectors):
        self.sectors = sectors
```

Ensuite, on implémente la fonction $E_{T_m(\alpha)}(p)$, qui projette une teinte p_{hue} sur la teinte la plus proche présente dans un des secteurs du template T décalé de l'angle α.

```python
def E(template, alpha, p): # p is an opencv hsv pixel. hue is half range

    best_hue = None
    best_dist = 99999
    for (wideness, angle_to_sector) in template.sectors:
        min_bound = (angle_to_sector + alpha - wideness / 2.0) % 360
        max_bound = (angle_to_sector + alpha + wideness / 2.0) % 360

        hue = get_hue(p)

        print(min_bound, max_bound, hue)

        if between(hue, min_bound, max_bound): return hue # if we're inside the sector

        dist = angle_distance(hue, min_bound)
        if dist < best_dist:
            best_dist = dist
            best_hue = min_bound

        dist = angle_distance(hue, max_bound)
        if dist < best_dist:
            best_dist = dist
            best_hue = max_bound

    return best_hue
```

Enfin, on peut définir la fonction $F(X, (T_m, \alpha))$, qui pour une image X, calcule une pseudo-distance entre l'harmonie de l'image et le template T_m décalé par l'angle α.

```
def F(X, template, alpha):
    acc = 0

    for i in range(0, X.height):
        for j in range(0, X.width):
            p = X[i, j]

            closest_hue = E(template, alpha, p)
            acc += arc_length(abs(get_hue(p) - closest_hue)) * p[1]/255.0
    return acc
```

Le problème de cette méthode est qu'elle doit itérer sur tous les pixels, donc elle est très longue en python. Deux pistes sont envisagées pour la suite: soit tenter de réécrire tous ces calculs comme une suite d'opérations sur des arrays numpy, soit utiliser Cython pour compiler la fonction F en C et l'importer comme une librairie python pour bénéficier de la vitesse du C.

Partie 9 : Tan et al, harmonisation de la palette

On a en entrée notre palette et la liste des poids qui permet d'exprimer l'image originale en fonction de la palette.

On veut maintenant harmoniser la palette, c'est à dire ajuster les teintes de toutes les couleurs pour qu'elles respectent une règle esthétique donnée (comme par exemple des couleurs complémentaires ou analogues). Cette harmonisation se fait dans l'espace LCh qui permet de séparer la luminosité, la chroma et la teinte.

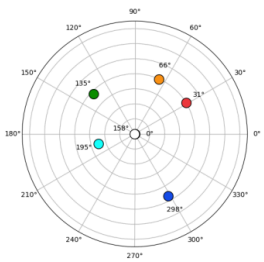
Pour adapter la palette au template choisi, on cherche les angles optimaux qui minimisent l'écart entre la teinte d'origine de chaque couleur et l'axe de teinte le plus proche défini par le template.

- Pour un template à un degré de liberté (exemple : monochromatique, complémentaire, etc.), on teste pour chaque angle de 0 à 359° et on calcule une mesure d'erreur (pondérée par la luminosité et la chroma) qui représente l'écart moyen entre les teintes de la palette et celles du template.
- Pour des templates à deux degrés de liberté (exemple : split ou double split), on parcourt une plage d'angles pour α_1 et α_2 (par exemple, α_2 de −30 à 30°) pour trouver la combinaison qui minimise cette erreur.

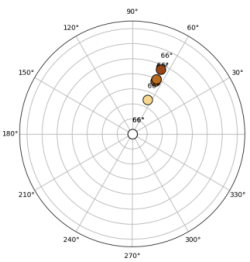
Ensuite on ajuste la palette : Pour chaque couleur de la palette en LCh, on détermine l'axe de teinte (issu du template) le plus proche de sa teinte actuelle et on force alors la couleur à adopter exactement cette teinte, tout en conservant sa luminosité et sa chroma.

Après cette harmonisation en LCh, on reconvertit les couleurs harmonisées en RGB pour obtenir la nouvelle palette.

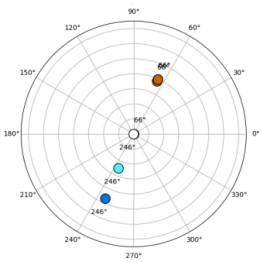
Exemple des harmonies avec plot sur cercle polaire :



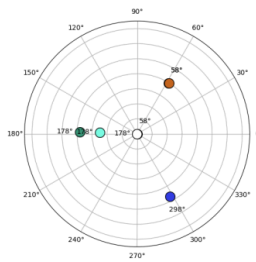
Palette Originale



Monochrome



Complémentaire



Triadique

Exemple d'images harmonisées :