Compte rendu 3

Traitement harmonique des images

Partie 5 : graphique de teinte

Afin de visualiser les effets de nos algorithmes de manière quantitative et intuitive, on veut exporter et visualiser un histogramme de la teinte de l'image.

Pour la partie export, c'est relativement facile. On convertit en hsv, puis on exporte uniquement le canal de la teinte en tant qu'histogramme normalisé.

```
// dans la classe ImageRGB
void saveHue(string name) const{
    auto Img = convertTo(PixelType::HSV);
    Img.apply([](const Vec3 & v){ // un peu bourrin pour isoler la teinte mais fonctionne
        return Vec3(v[0]);
      }
    ).saveHistogramProbabilisticData(name, true);
}
```

On obtient un fichier d'extension *.hdat*, qui contient le tableau des valeurs voulues.

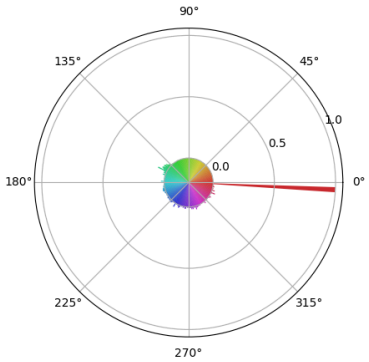
Ensuite, le code python suivant, ajouté au fichier convert.py, permet de convertir tous les hdats d'un dossier en un graphique de teinte similaire à ceux présents dans certains des papiers étudiés.

```
def hue_plot(path):
    x_plot, p1 = [], []
    with open(path, 'r') as file:
        reader = csv.reader(file, delimiter=' ')
        line_len = None
        for line in reader:
            assert line_len is None or line_len == len(line)
            line_len = len(line)

            x_plot.append(float(line[0]))
            p1.append(float(line[1]))
    p1 = np.array(p1)
    p1 = np.log(1.0 + p1)
    p1 /=  p1.max()

    fig = plt.figure(num=1, clear=True)
    N = 256
    theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
    radii = 0.5 * p1 + 0.1
    width = 2*np.pi / 256 + 0.01
    ax = plt.subplot(projection='polar')
    ax.set_yticks([0.0, 0.5, 0.75, 1.0], ['', '0.0', '0.5', '1.0'])
    ax.bar(theta, radii, width=width, bottom=0.4, color=colors, alpha=1.0)
    return fig
```

Et on obtient les résultats suivants



histogramme original de l'image de test

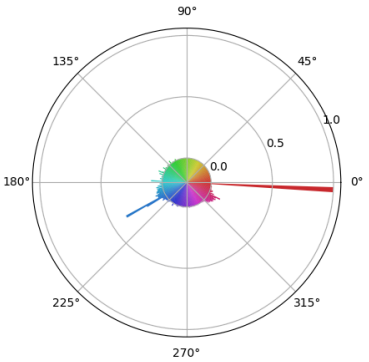


image avec teinte décalée
Il semblerait que mes valeurs soient pas bonnes… je règlerai le problème pour la prochaine fois.

Partie 6 : Tan et al, suite

Maintenant qu'on a une palette qui représente notre image, l'étape suivante consiste à découper l'image pour pouvoir l'exprimer en fonction de cette palette. C'est à dire qu'on veut pouvoir exprimer chaque pixel de l'image par la somme pondérée des couleurs de la palette.

Petit twist : pour cette partie, l'article propose d'utiliser l'espace RGBXY au lieu de l'espace RGB pur. En ajoutant les coordonnées (X, Y), on obtient une décomposition qui reste cohérente spatialement. Autrement dit, des pixels proches dans l'image et de couleur similaire se verront attribuer des poids proches, ce qui évite les artefacts de « speckling » (petits points isolés) lors de la reconstruction.

Pour découper l'image en couches, voici les étapes principales : • Calcul de l'enveloppe convexe dans l'espace RGBXY • Tessellation de l'enveloppe Pour obtenir des poids de mélange, on découpe (ou tesselle) cette enveloppe convexe en plusieurs tétraèdres en utilisant une triangulation Delaunay. • Calcul des coordonnées barycentriques Pour un pixel situé dans un tétraèdre, il existe une unique combinaison de poids qui exprime sa position relative aux sommets du tétraèdre. Ces poids, appelés coordonnées barycentriques, représentent la contribution de chacune des couleurs associées aux sommets pour reconstituer la couleur du pixel.

À la fin de la décomposition, chaque pixel se voit attribué une liste de poids de la même taille que le nombre de couleurs de la palette. En décomposant l'image par couleur de palette, ça donne :