Compte rendu 4

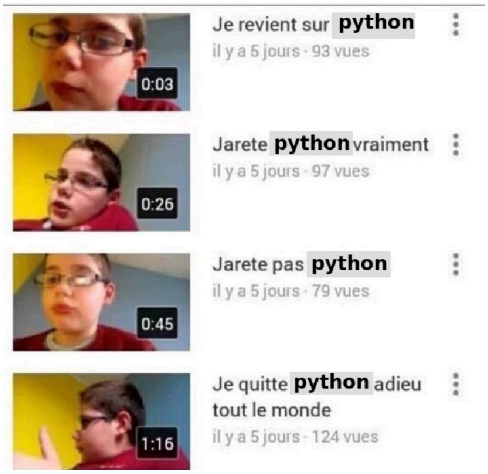
Traitement harmonique des images

Partie 10 : Cohen-or et al, Pas si vite!

Dans l'épisode précédent, nous expliquions pourquoi nous étions passés sur python.

Cependant, l'extrême lenteur du langage, qui avait été soulevée alors, fut catastrophique: En effet, l'algorithme classique de Cohen-Or et al (sans les optimisations pour les couleurs opposées), sur une image et une machine données, prend plus de **trois minutes** à terminer en python, contre moins de **dix secondes** (sans multithreading) en C++.

Comment on le sait? On a tout recodé en C++.



Voici donc quelques premiers résultats:

Image de base

Image traitée
(meilleur angle + template)

Image traitée
(meilleur angle, template « I »)

On remarque les effets du tearing
sur le mur de droite.

L'algorithme semble fonctionner, quoi que nos résultats semblent quelques peu différentes de l'autre groupe. Nous allons cross-examiner nos résultats avec eux pour trouver les causes des problèmes.

Partie 11 : Tan et al, Interface utilisateur

On a fini l'implémentation de l'algorithme de Tan et Al et elle fonctionne correctement.

L'étape suivante consiste à développer une interface permettant plusieurs actions : harmoniser une image, l'exporter, ainsi qu'enregistrer les palettes (complète et simplifiée) et les différentes couches issues de la décomposition RGBXY. Concernant l'harmonisation, l'utilisateur doit pouvoir connaître le meilleur pattern donné par l'algorithme, tout en ayant la possibilité d'en utiliser d'autres.

Étant donné que l'algorithme a été codé en python, on a ajouté un serveur flask au projet afin de créer une interface web. Le programme principal met en place un serveur HTTP et un serveur de WebSocket.

Lorsqu'un utilisateur accède au site, une connexion WebSocket est établie et sera utilisée pour toutes les interactions avec le serveur.

Voici un aperçu de l'interface :