

Compte rendu 2

Traitement harmonique des images

Partie 3 : Améliorations à notre librairie C++	1
Partie 4 : Tan et al	2

Partie 3 : Améliorations à notre librairie C++

Afin de faciliter l'utilisation des fonctionnalités, nous avons modifié notre importateur et exportateur d'images pour pouvoir charger et sauvegarder aux formats jpeg et png.

Pour cela, nous utilisons les librairies `stb_image` et `stb_image_write`, qui prennent en charge le parsing et nous renvoient la donnée sous forme de liste d'octets.

On fait l'interface avec notre classe **ImageRGB** en utilisant la classe utilitaire **ImageByte**. Au lieu de vecteurs 3, cette classe représente les couleurs par une classe **Color**, qui contient un triplet d'octets et les fonctions de conversions nécessaires avec les **Vec3**.

Donc pour charger une image, il suffit donc de faire cela.

```
static ImageRGB from(const string &name){
    int w, h;
    int comp = -1;

    unsigned char* image = stbi_load(name.c_str(), &w, &h, &comp, 3);

    ImageRGB res(w, h);

    if (comp == 1){
        for (int i = 0; i < w * h * comp; i+=comp){
            res.data[i] = Vec3(image[i]/255.0);
        }
    }
    else if (comp == 3 || comp == 4){
        for (int i = 0; i < w * h; i+=1){
            res.data[i] = Vec3(image[3*i]/255.0, image[3*i+1]/255.0, image[3*i+2]/255.0);
        }
    }
    else {
        std::cout << "[IMAGE LOADING] number of components (" << comp << ") is not 1, 3 or 4." << std::endl;
        exit(1);
    }

    stbi_image_free(image);

    return res;
}
```

Et pour sauvegarder, après conversion en **ImageByte**, de faire ceci.

```
void ImageByte::saveAs(string name, float quality) const{
    string ext = name.substr(name.find_last_of("."));

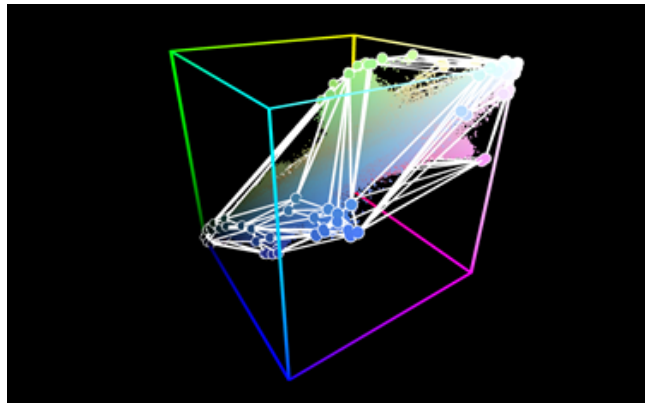
    char * data_start = (char * )data.data();

    if (ext == ".pgm" || ext == ".ppm") saveAsPPM(name, w, h, data); // our old function
    else if (ext == ".jpg" || ext == ".jpeg") stbi_write_jpg(name.c_str(), w, h, 3,
data_start, 100 * quality);
    else {
        if (! (ext == ".png")) name += ".png";
        stbi_write_png(name.c_str(), w, h, 3, data_start, w * 3);
    }
}
```

Partie 4 : Tan et al

Dans la méthode présentée par cet article, l'harmonisation d'une image se fait en deux étapes : une phase de prétraitement où l'on calcule la palette de l'image puis détermine les poids de chaque couleur pour chaque pixel, et une étape de traitement.

Cette semaine j'ai implémenté l'étape de calcul de palette. Pour trouver cette palette, on calcule d'abord l'enveloppe convexe des pixels, ce qui donne :



Ensuite, on fusionne itérativement des arêtes. Pour chaque arête candidate, on résout un problème d'optimisation linéaire (LP) qui permet de trouver le point de fusion qui minimise l'ajout de volume, c'est-à-dire l'erreur induite par la fusion. • L'erreur est évaluée via la RMSE, qui mesure la différence entre la palette initiale et celle simplifiée. • On choisit alors la fusion qui ajoute le moins de volume et on met à jour l'enveloppe convexe avec ce nouveau point.

Critères d'arrêt : L'étape de simplification se termine soit quand on atteint un nombre de sommets prédéfini, soit quand les itérations n'apportent plus d'amélioration significative (ou quand le nombre de sommets stagne).

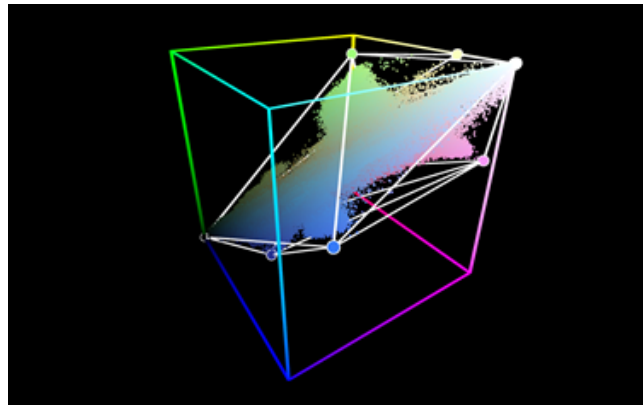



Image source :	Palette complète et simplifiée
	<div data-bbox="788 698 1145 745" data-label="Image"> </div> <div data-bbox="788 784 1145 831" data-label="Image"> </div>